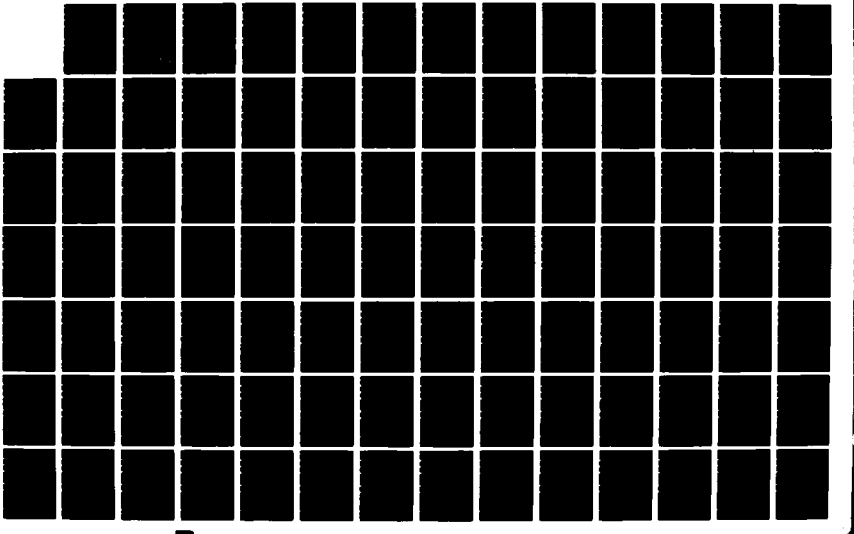
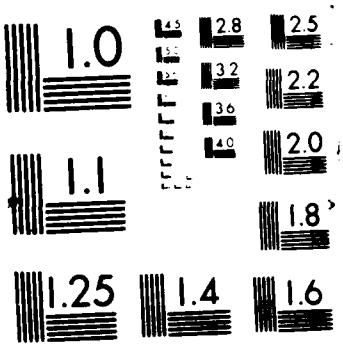
AD-A186 743 VOICE TECHNOLOGY USING PERSONAL COMPUTERS(U) AIR FORCE 1/3
INST OF TECH WRIGHT-PATTERSON AFB OH G L TALBOT 1987
AFIT/CI/NR-87-43T

UNCLASSIFIED F/G 25/4 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

VOICE TECHNOLOGY

USING

PERSONAL COMPUTERS

by

Gary L. Talbot

A Report Submitted in Partial Fulfillment
of the Requirement for the Degree
of Master of Science
(Management Information Systems)
in The University of Arizona
1987

Master Committee:
Dr. Doug Vogel
Mr. Bill Saints
Ms. Kimlynn Middleton

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 87-43T	2. GOVT ACCESSION NO. AD-A186743	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Voice Technology Using Personal Computers		5. TYPE OF REPORT & PERIOD COVERED THESIS/DISSERTATION
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gary L. Talbot		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of Arizona		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433-6583		12. REPORT DATE 1987 13. NUMBER OF PAGES 263
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

Lynn Wolaver 13 May 87

LYNN E. WOLAVER
Dean for Research and
Professional Development
AFIT/NR

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

VOICE TECHNOLOGY USING PERSONAL COMPUTERS
by
Gary L. Talbot, Capt, USAF

A Report Submitted in Partial Fulfillment
of the Requirement for the Degree
of Master of Science
(Management Information Systems)
in the University of Arizona
1987

263 pages

My software project had two objectives in mind, thus my report is divided into two parts.

The first objective and subject of Part I was to develop a Turbo Pascal procedure to drive the **IBM Voice Communications Application Program Interface** software which interfaces the **IBM Voice Communications Adapter** hardware to synthesize speech from text. The resulting Turbo Pascal procedure, **SPEAK.INC**, was designed to allow any user the ability to produce speech-from-text from within any Turbo Pascal program. In addition, three application programs that can be applied for introductions, explanations, error messages, etc. were developed using the procedure. **SAY.COM** allows a user the ability to produce speech from the command line or from within a batch file. **SAYTEXT.COM** verbalizes text from within any text file. **REMIND.COM** is a memory-resident program that produces verbal messages at preprogrammed times. User guides and system documentation guides for the procedure and the three application programs are found in Part I, Chapter One through Chapter Four.

The second objective, covered in Part II, explores voice recognition tools through the **IBM Voice-Activated Keyboard Utility**. This utility allows user-defined applications to be built that free the user from the keyboard during interaction with group members in discussions, presentations, etc. or in any situation where the user requires mobility from the keyboard. In Chapter Five, a user guide with an example user application is provided to assist setting up applications incorporating the utility to use voice recognition.

ABSTRACT

VOICE TECHNOLOGY USING PERSONAL COMPUTERS

by

Gary L. Talbot

Chairman: Dr. Doug Vogel

My software project had two objectives in mind, thus my report is divided into two parts.

The first objective and subject of Part I was to develop a Turbo Pascal procedure to drive the **IBM Voice Communications Application Program Interface** software which interfaces the **IBM Voice Communications Adapter** hardware to synthesize speech from text. The resulting Turbo Pascal procedure, **SPEAK.INC**, was designed to allow any user the ability to produce speech-from-text from within any Turbo Pascal program. In addition, three application programs that can be applied for introductions, explanations, error messages, etc. were developed using the procedure. **SAY.COM** allows a user the ability to produce speech from the command line or from within a batch file. **SAYTEXT.COM** verbalizes text from within any text file. **REMIND.COM** is a memory-resident program that produces verbal messages at preprogrammed times. User guides and system

documentation guides for the procedure and the three application programs are found in Part I, Chapter One through Chapter Four.

The second objective, covered in Part II, explores voice recognition tools through the **IBM Voice-Activated Keyboard Utility**. This utility allows user-defined applications to be built that free the user from the keyboard during interaction with group members in discussions, presentations, etc. or in any situation where the user requires mobility from the keyboard. In Chapter Five, a user guide with an example user application is provided to assist setting up applications incorporating the utility to use voice recognition.

VOICE TECHNOLOGY

USING

PERSONAL COMPUTERS

by

Gary L. Talbot

A Report Submitted in Partial Fulfillment
of the Requirement for the Degree
of Master of Science
(Management Information Systems)
in The University of Arizona
1987

Master Committee:
Dr. Doug Vogel
Mr. Bill Saints
Ms. Kimlynn Middleton

ACKNOWLEDGMENTS

I wish to express my most sincere thanks and appreciation to Ms. Kimlynn Middleton, Ms. Kendall Cliff, and Mr. Bill Saints for their time and support in helping me accomplish this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PART I

TEXT-TO-SPEECH INTERFACE

and

APPLICATIONS

CHAPTER 1

TEXT-TO-SPEECH INTERFACE

USER GUIDE

Text-to-Speech Interface

for the

IBM Voice Communications Adapter

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

The **SPEAK** procedure is written as an interface between Turbo Pascal programs and the **IBM Voice Communications Adapter** and allows the calling Turbo Pascal program to produce intelligible speech from an ASCII text string.

This procedure was written using Turbo Pascal, Version 3.0, to execute commands of the **IBM Voice Communications Application Program Interface (VC API)** software or the **Voice Communications Operating System** software that drives the **IBM Voice Communications Adapter**. This hardware device consists of a specialized micro processor, memory, and supporting hardware which allows text to be translated to spoken language. The **VC API** software is called from the **SPEAK** procedure by setting up registers and supporting parameter blocks and then executing a DOS interrupt 14H.

To produce speech, certain base commands must be executed to provide initialization, termination, and resource management. The text-to-speech function set produces text-to-speech translation and is loaded into the micro processor of the **Voice Communications Adapter** to execute commands producing the actual text-to-speech translation.

Basic actions that are accomplished by the **SPEAK** procedure are:

-obtain software resources using the open command;

-claim hardware resources using the claim hardware command;

-connect devices to appropriate port using the connect devices to port command;

-load the text-to-speech function set onto the hardware and connect it to the appropriate port using the connect function to port command;

-initialize the speech function;

-produce the actual text-to-speech translation;

-release all resources via the close command.

Each command requires different register and parameter settings before the interrupt 14H is issued. On return, each command will return command specific error codes if an error occurs.

The **SPEAK** procedure requires three parameters to be passed when it is called. The first is the name of a string containing a sentence of up to 240 ASCII text characters that is followed by an end of sentence terminator, either a period (.), a question mark (?), or an exclamation point (!), that the user wishes to be

translated into spoken language. The second parameter is an integer specifying the desired voice pitch. The third and final parameter is also an integer that specifies the desired rate of speech.

This guide is intended to simplify the task of a user wishing to produce speech from text from within a Turbo Pascal program. The sections following will discuss hardware and software requirements that are necessary to use this procedure. Also, installation instructions for using the speech-to-text procedure will be discussed. An overview of using the **SPEAK** routine is covered under the operating instructions section and an example is provided. Next, input and output formats and descriptions are discussed. Finally, references for additional information are given.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware:

Minimum hardware requirements are given in Table 1.1.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-sided diskette drive (360 KB / 1.2 MB) and one fixed disk
- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector

Table 1.1

Software:

Minimum software requirements are given in Table 1.2.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0

Table 1.2

INSTALLATION INSTRUCTIONS

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette *named vcapi.* The **Voice Communications Operating Subsystem Program** diskette is self-installing and is a

fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the *autoexec.bat* file:

```
set vcapi = y:\vcapi\
```

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

```
y:\vcapi\vcapidrv /o 10
```

(the /o 10 option allows the text-to-speech function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

OPERATING INSTRUCTIONS

Procedure **SPEAK** serves as an interface to the **IBM Voice Communications Applications Program Interface** for the text-to-speech (speech synthesis) function set. It can be included within any Turbo Pascal program in which the user wishes to have a passage of ASCII text translated to

speech. The only required lines within the calling program are:

- a type declaration,
- an include statement to include the **SPEAK** procedure, and
- the call to the procedure.

Parameters that must be passed to the SPEAK procedure are:

- the name of the string containing up to 240 ASCII text characters forming a sentence and followed by a sentence terminator, either a period (.), a question mark (?), or an exclamation point (!).;
- an integer, p, giving the baseline pitch; and
- an integer, r, which sets the rate of speech.

Pitch: the baseline pitch must be set to 0 or be an integer in the range between 50 and 100, inclusive. Any other value supplied that does not meet these requirements will be defaulted to the normal pitch rate of 85. The special pitch of 0 will produce a whispering voice. Pitch may be adjusted dynamically by changing this parameter within any call to the **SPEAK** procedure at any time. Resolution of baseline pitch is about 10 so differences such as 103 and 107 may not be detectable. Higher numbers produce higher pitches.

Rate: the range for the speech rate parameter is between 50 and 250, inclusive. Any integer not supplied within this range will default to the normal speech rate of 150 words per minute. Again, maximum resolution is about 10 words per minute so values such as 123 and 130 may not be detectable. Speech rate is also adjustable by changing the parameter passed in the call to the **SPEAK** procedure. Higher values produce faster rates of speech.

When using the **SPEAK** procedure within a Turbo Pascal program, the user may wish to include the compiler option {$V-} to relax checking of the length of strings passed to the **SPEAK** procedure. That is, a string with length of 80, 128, etc. up to the maximum allowable length of 240 characters may be passed as the parameter to the procedure. ** Note ** A string of type 'longstr' for the VAR parameter definition used by the **SPEAK** procedure is still required to be defined.

An example user program follows:

```
Program  Calling_Program;

{$V-}                    {optional compiler directive to
                          relax parameter string checking}
```

```
type      longstr  = string[240]    {this must be
                                     supplied since it
                                     is declared as a
                                     VAR parameter in
                                     SPEAK}

{other user declarations, variables, functions, procedures,
etc.}

{$I speak.inc}          {includes the SPEAK procedure}

begin                   {user program starts here}

   {user code to set up the string to pass to the
    SPEAK procedure goes here}

   speak(stringname,p,r);  {call the SPEAK procedure
                            to say the text in the
                            string designated by
                            stringname at pitch, p,
                            and at rate, r.}

     {an example of a real call might look like this:

                speak(textbuf,85,150);    }
```

```
{more user code if desired}

end.                        {end of user program}
```

Error codes are returned directly by the **SPEAK** procedure to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 1.3. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

ERROR CODES RETURNED BY SPEAK

Command	Error Code	Explanation
Open	0	Successful
	2	API inoperative
	16	RCB not available
	64	Invalid card number
Claimhdw	0	Successful
	2	Card Inoperative
	4	RCB invalid or not pen

(cont.)

	16	At least one resource seized
	32	Unsupported hardware
	48	Combination of 16 and 32 above
Conndtop	0	Successful
	2	Card inoperative
	4	RCB invalid or not open
	16	Port or devices not claimed
	32	Unsupported devices
	64	Unsupported connection
Connftop	0	Successful
	2	Card inoperative
	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency

(cont.)

	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped
Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided
	128	No null found in text
	256	Pause pending (must call resume first)
Close	0	Successful
	2	Card inoperative
	4	RCB invalid or not open

Table 1.3

If a return code other than 0 is returned when a command is executed by the driver, then a message is written to the console telling which command returned the error and which error code was returned. The only ones which a user might see are error codes 16 or 64 from the speak command. This usually indicates that the end of sentence punctuation was not provided when the **SPEAK** procedure was called. A programmer may desire that the procedure not notify the user if an error code is returned and this logic is easily deleted from the procedure. It is basically incorporated as an aid in development when first installing the speech-to-text software and hardware to track down errors that might occur. It can also be a helpful reminder to the user to end sentences with terminator punctuation, although user applications can be written to default to a period, etc., if no punctuation is provided.

INPUT AND OUTPUT FORMATS AND DESCRIPTIONS

Input to the **SPEAK** procedure is in the form of strings consisting of up to 240 ASCII text characters forming sentences and followed by a sentence terminator (period (.), question mark (?), or an exclamation point (!)).

Additional inputs are two integer values, the first providing the desired base pitch and the second setting the desired rate of speech. Output is in the form of intelligible speech. Other output is written error messages to the user console.

PROGRAM LISTING

```
{************************************************************
Procedure Speak serves as an interface to the IBM Voice
Communications Applications Program Interface for the Text-
to-Speech (speech synthesis) function set. It can be
included in any Turbo Pascal program in which the user
wishes to have a passage of text spoken. The only required
lines within the calling program are a type declaration, an
include statement to include the procedure, and the call
to the procedure. Parameters that must be passed to the
speak procedure are the name of a string containing a
sentence of up to 240 ASCII text characters that ends with
a sentence terminator('.','?',or '!'), an integer, p,
giving the baseline pitch (the range for pitch is 0 or
between 50 and 100), and an integer, r, which sets the
speech rate (the range for speech rate, r, is between 50
and 250). A pitch of 0 will produce a whispering voice
while other values not between 50 and 200 will default to
the normal pitch rate of 85. Pitch may be adjusted at any
time by replacing this value and speach will remain at this
same pitch until another value is input. Resolution of
baseline pitch is about 10 so differences such as 103 and
111 may not be detectable. Higher numbers produce higher
pitches. If a value outside the range of 50 and 100 is not
used with the speech rate, r, then the default is to the
normal rate of 150 words per minute. Again, maximum
resolution is about 10 words per minute so values such as
123 and 127 may not be detected. Speech rate is also
adjustable by changing the value passed and this rate
remains in effect until a different value is supplied.
Higher values produce faster rates of speech.
```

When using Turbo Pascal, the compiler option {$V-} may be used to relax checking of the length of the buffer passed to the speak procedure. That is, a buffer with length of 80, 128, etc may be passed. However, it is still required to define a string of type 'longstr' for the var parameter used in the speak procedure.

An example user program follows:

```
*********************************************************

Program Calling_Program;

{$V-}              {optional compiler directive to relax
                    parameter length checking}
type   longstr = string[240];   {this must be supplied
                                  since it is declared
                                  as a Var parameter in
                                  Speak}

     {other user declarations, variables,
             functions, procedures, etc.}


{$I speak.inc}     {includes the speak interface code}

begin              {user program starts here}

{user code to set up a buffer to send to speak
   procedure}

speak(string,p,r)   {call the speak procedure to speak
                       text in the string at pitch, p,
                       and at rate, r}

{more user code if desired}

end.                  {end of user program}

*********************************************************
```

Reference: *IBM Voice Communications Application Program Interface Reference Vol 1 & 2*

For additional information on error codes returned, see the above reference or consult the *Text-to-Speech Interface for the IBM Voice Communication Adapter Guide, Talbot, Summer, 1987.*

```
*********************************************************}
```

```
Procedure Speak(Var talk:longstr; p,r: integer);

label loop,99,fini;

type  result   = record
                   ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
                 end;

      plist    = array[0..5] of integer;

      shortstr = string[16];

var   reg    :result;          {record type to call
                                interrupt}
      rcb    :integer;         {storage for the resource
                                control block}
      bid    :integer;         {storage for the base id}
      cid    :integer;         {storage for partition 2
                                connection id}
      pb     :plist;           {the parameter block}
      k      :integer;         {length of text}
      pitch  :string[3];       {voice pitch string}
      rate   :string[3];       {voice rate string}
      setbuf :shortstr;        {set pitch and rate
                                buffer}

begin

{setup pitch, p, and speech rate, r}

if p in [0,50..200]
   then str(p,pitch)           {convert pitch to string}
   else pitch:= '85';          {default to normal pitch}

if r in [50,250]
   then str(r,rate)            {convert rate to string}
   else rate:= '150';          {default to normal rate}

setbuf:=^[+'['+pitch+'p'+^[+'['+rate+'r'+^@;
                    {setup the pitch and rate buffer}


{open command to obtain a resource control block and
 connection ids}

reg.ax:=$1111;                 {function code for open
                                command}
reg.dx:=$021f;                 {board I/O address}
reg.es:=seg(pb);               {parameter block segment}
reg.bx:=ofs(pb);               {parameter block offset}
intr($14,reg);                 {call interrupt 14}
```

```
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in open.');
     goto 99;
   end;
rcb:=pb[1];                     {save resource control
                                 block}
bid:=pb[2];                     {save base id}
cid:=pb[4];                     {save partition 2
                                 connection id}

{claim h/w resources for the rcb using claimhdw
   command}

reg.ax:=$111a;                  {function code for
                                 claimhdw command}
reg.dx:=bid;                    {need base id in dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[2]:=$2602;                   {claim port 2, partition
                                 2, spkr, microphone}
pb[3]:=$0000;                   {no base interrupt}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in claim.');
     goto 99;
   end;

{connect devices to the port using conndtop command}

reg.ax:=$1121;                  {function code for
                                 conndtop}
reg.dx:=bid;                    {need base id in dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[2]:=2;                       {connect to port 2}
pb[3]:=$0600;                   {connect microphone and
                                 speaker}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in connect devices.');
     goto 99;
   end;
```

```
{load function set into a port and connect it using
 the connftop command}

reg.ax:=$111f;                  {function code for
                                 connftop command}
reg.dx:=bid;                    {need base id in dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[1]:=cid;                     {need cid in the
                                 parameter block}
pb[2]:=2;                       {connect to port 2}
pb[3]:=10;                      {connect text-to-speech
                                 function}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in connect
              function.');
     goto 99;
   end;

{the initialize text-to-speech function set data
   structures}

reg.ax:=$1113;                  {function code for
                                 initialize data
                                 structures}
reg.dx:=cid;                    {need connection id in
                                 dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[1]:=cid;                     {need cid in parameter
                                 block also}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in initialize speech
              function.');
     goto 99;
   end;

{the text-to-speech speak command}

{set the pitch and rate by outputting setbuf}

reg.ax:=$111e;                  {function code for speak
                                 command}
reg.dx:=cid;                    {need connection id in
                                 dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
```

```
pb[1]:=cid;                          {need cid in parameter
                                      block also}
pb[2]:=2;                            {32 bit address for
                                      buffer setbuf}
pb[3]:=ofs(setbuf)+1;                {setbuf address offset,
                                      offset 1 for length}
pb[4]:=seg(setbuf);                  {setbuf address segment}
intr($14,reg);                       {call interrupt 14}
if pb[0] <> 0                        {zero if no error}
   then
      begin
        writeln('An error occurred in speech
                 function.');
        goto 99;
      end;

{say the text line that was passed as a parameter}

reg.ax:=$111e;                       {function code for speak
                                      command}
reg.dx:=cid;                         {need connection id in
                                      dx}
reg.es:=seg(pb);                     {parameter block segment}
reg.bx:=ofs(pb);                     {parameter block offset}
pb[1]:=cid;                          {need cid in parameter
                                      block also}
pb[2]:=2;                            {32 bit address for
                                      buffer talk}
k:=length(talk);                     {find the length of the
                                      buffer}
talk[k+1]:=^[;                       {put in an ESC}
talk[k+2]:='[';                      {and a left bracket}
talk[k+3]:='i';                      {and an i to create
                                      interrupt}
talk[k+4]:=^@;                       {add a null at the end}
pb[3]:=ofs(talk)+1;                  {use the buffer passed in
                                      talk, offset 1 for
                                      length}
pb[4]:=seg(talk);                    {segment for talk}
intr($14,reg);                       {call interrupt 14}
if pb[0] <> 0                        {zero if no error}
   then
      begin
        writeln('An error occurred in speech
                 function.');
        goto 99;
      end;
goto fini;

99:  writeln('Return Code is ',pb[0]);  {tell the user
                                         what code was
                                         returned}
```

```
{close command to release resources}

fini:                                {come here always to
                                      release resources}
reg.ax:=$1112;                       {function code for close}
reg.dx:=bid;                         {need base id in dx}
reg.es:=seg(pb);                     {parameter block segment}
reg.bx:=ofs(pb);                     {parameter block offset}
pb[1]:=rcb;                          {resource control block
                                      to release resources}
intr($14,reg);                       {call interrupt 14}
if pb[0] <> 0 then                   {zero if no error}
   begin
     writeln('An error occurred in close.');
     writeln('Return Code is ',pb[0]);  {tell the user
                                         what code was
                                         returned}
   end;

end;                                 {procedure speak}
```

REFERENCES

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

CHAPTER 2

PROGRAM SAY.COM

Program SAY.COM

User Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

SAY is a program written in Turbo Pascal, Version 3.0, that allows a user to input text that (s)he wishes translated into intelligible spoken language. The program provides a quick and easy way of producing speech from a text string entered from the DOS command line.

For example, within a batch file, or at any other time when control is at the DOS command line, if the user wishes the sentence, "Please enter your name now.", to be spoken, then all (s)he has to enter is: **say Please enter your name now.** This command will activate the **SAY** program and cause the text that is passed on the command line as parameters to be voiced over a speaker. A user must assure that the **IBM Voice Communications Adapter** (hardware) and the **IBM Voice Communications Operating System** (software) are installed on the machine at which they are working for the **SAY** program to work correctly.

Text-to-speech translation is made possible by use of the **SPEAK** procedure which serves as an interface to the **IBM Voice Communications Operating System** which in turn drives the **IBM Voice Communications Adapter** that produces intelligible speech from text.

This guide is intended to simplify the task of a user wishing to produce speech from text from the DOS command line. The sections following will discuss hardware and software requirements that are necessary to use this program. Also, installation instructions for using the **SAY** program will be discussed. An overview of using the **SAY** command is covered under the operating instructions section and examples are provided. Next, input and output formats and descriptions are discussed. Finally, references for additional information are provided.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware:

Minimum hardware requirements are given in Table 2.1.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-

(cont.)

sided diskette drive (360 KB / 1.2 MB) and one fixed disk

- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector)

Table 2.1

Software:

Minimum software requirements are given in Table 2.2.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0 (for compilation purposes only)
- SAY.COM

Table 2.2

INSTALLATION INSTRUCTIONS

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette named *vcapi*. The **Voice Communications Operating Subsystem Program** diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the **autoexec.bat** file:

set vcapi = y:\vcapi

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

y:\vcapi\vcapidrv /o 10

(the /o 10 option allows the text-to-speech function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

OPERATING INSTRUCTIONS

The **SAY** program will produce intelligible speech from the text entered as parameters on the command line. Input is limited to 127 total characters due to Turbo Pascal limitations. Text to be spoken should be followed with the desired end of sentence terminator, either a period (.), a question mark (?), or an exclamation point (!). If the user forgets to provide the end of sentence terminator, then the default is a period. Sentence intonation varies according to the terminator provided. Pitch and rate of speech are set at 65 and 170 words per minute within the **SAY** program. If the user desires to change these rates, (s)he may change the parameters passed in the call to the

SPEAK procedure within the program and recompile the program. When recompiling, make sure that the option to produce a .COM file has been selected from within Turbo Pascal. For additional information on the **SPEAK** procedure and pitch and rate, see reference 3 given at the end of this guide.

Example uses of the SAY program follow:

A>say This is a mighty fine computer system!

A>say Do you want to delete all files?

A>say It is now time to have a coffee break.

Error codes may be returned directly by the **SAY** command to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 2.3. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

ERROR CODES

Command	Error Code	Explanation
Open	0	Successful
	2	API inoperative
	16	RCB not available
	64	Invalid card number
Claimhdw	0	Successful
	2	Card Inoperative
	4	RCB invalid or not open
	16	At least one resource seized
	32	Unsupported hardware
	48	Combination of 16 and 32 above
Conndtop	0	Successful
	2	Card inoperative
	4	RCB invalid or not open
	16	Port or devices not claimed

(cont.)

	32	Unsupported devices
	64	Unsupported connection
Connftop	0	Successful
	2	Card inoperative
	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency
	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped

(cont.)

Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided
	128	No null found in text
	256	Pause pending (must call resume first)
Close	0	Successful
	2	Card inoperative
	4	RCB invalid or not open

Table 2.3

If a return code other than 0 is returned when a command within the **SPEAK** procedure is executed by the driver, then a message is written to the console telling which command returned the error and which error code was returned. The only ones which a user might see are error

codes 16 or 64 from the speak command. This usually indicates that the end of sentence punctuation was not provided when the **SPEAK** procedure was called. A programmer may desire that the procedure not notify the user if an error code is returned and this logic is easily deleted from the **SPEAK** procedure. It is basically incorporated as an aid in development when first installing the speech-to-text software and hardware to track down errors that might occur.

INPUT AND OUTPUT FORMATS AND DESCRIPTIONS

*Input to the **SAY** program is a text string of up to 127* characters that form a logically complete sentence that ends with a sentence terminator, either a period (.), a question mark (?), or an exclamation point (!).

Output is in the form of intelligible spoken translation of the text that is input. Other output is in the form of error messages directly to the user console.

REFERENCES

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

3. *Text-to-Speech Interface for the IBM Voice Communications Adapter, Talbot, Summer 1987*

Program SAY.COM

System Documentation Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

SAY is a program written in Turbo Pascal, Version 3.0, that allows a user to input text that (s)he wishes translated into intelligible spoken language. The program provides a quick and easy way of producing speech from a text string entered from the DOS command line.

For example, within a batch file, or at any other time when control is at the DOS command line, if the user wishes the sentence, "Please enter your name now.", to be spoken, then all (s)he has to enter is: **say Please enter your name now.** This command will activate the **SAY** program and cause the text that is passed on the command line as parameters to be voiced over a speaker. A user must assure that the **IBM Voice Communications Adapter** (hardware) and the **IBM Voice Communications Operating System** (software) are installed on the machine at which they are working for the **SAY** program to work correctly.

Text-to-speech translation is made possible by use of the **SPEAK** procedure which serves as an interface to the **IBM Voice Communications Operating System** which in turn drives the **IBM Voice Communications Adapter** that produces intelligible speech from text.

SYSTEM SPECIFICATIONS

Both the **IBM Voice Communications Operating System** and the **IBM Voice Communications Application Program Interface** software are required for operation of the **SAY.COM** program. Details on installation are provided below.

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **IBM Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette named *vcapi*. The **Voice Communications Operating Subsystem Program** diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the **autoexec.bat** file:

set vcapi = y:\vcapi

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

y:\vcapi\vcapidrv /o 10

(the /o 10 option allows the text-to-speech function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

HARDWARE REQUIREMENTS

Minimum hardware requirements are given in Table 2.4.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double -sided diskette drive (360 KB / 1.2 MB) and one fixed disk

(cont.)

- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector)

Table 2.4

SOFTWARE REQUIREMENTS

Minimum software requirements are given in Table 2.5.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0 (for compilation only)
- SAY.COM

Table 2.5

DESIGN DETAILS

The **SAY.COM** program uses the **SPEAK** procedure (see references) to interface the **IBM Voice Communications Interface Program** to issue commands to the **IBM Voice Communications Adapter**. Words are passed as parameters on the command line and are placed into a string buffer which is passed to the **SPEAK** procedure to be translated into speech.

IMPLEMENTATION DETAILS

The **SAY** program will produce intelligible speech from the text entered as parameters on the command line. Input is limited to 127 total characters due to Turbo Pascal limitations. Text to be spoken should be followed with the desired end of sentence terminator, either a period (.), a question mark (?), or an exclamation point (!). If the user forgets to provide the end of sentence terminator, then the default is a period. Sentence intonation varies according to the terminator provided. Pitch and rate of speech are set at 65 and 170 words per minute within the **SAY** program. If the user desires to change these rates, (s)he may change the parameters passed in the call to the **SPEAK** procedure within the program and recompile the

program. When recompiling, make sure that the option to produce a .COM file has been selected from within Turbo Pascal. For additional information on the **SPEAK** procedure and pitch and rate, see reference three given at the end of this guide.

Example uses of the **SAY** program follow:

A>say This is a mighty fine computer system!
A>say Do you want to delete all files?
A>say It is now time to have a coffee break.

Error codes may be returned directly by the **SAY** command to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 2.6. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

ERROR CODES

Command	Error Code	Explanation
Open	0	Successful
	2	API inoperative
	16	RCB not available
	64	Invalid card number
Claimhdw	0	Successful
	2	Card Inoperative
	4	RCB invalid or not open
	16	At least one resource seized
	32	Unsupported hardware
	48	Combination of 16 and 32 above
Conndtop	0	Successful
	2	Card inoperative
	4	RCB invalid or not open
	16	Port or devices not claimed
	32	Unsupported devices
	64	Unsupported connection
Connftop	0	Successful
	2	Card inoperative

(cont.)

	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency
	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped
Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided

(cont.)

	128	No null found in text
	256	Pause pending (must call resume first)
Close	0	Successful
	2	Card inoperative
	4	RCB invalid or not open

Table 2.6

If a return code other than 0 is returned when a command within the **SPEAK** procedure is executed by the driver, then a message is written to the console telling which command returned the error and which error code was returned. The only ones which a user might see are error codes 16 or 64 from the speak command. This usually indicates that the end of sentence punctuation was not provided when the **SPEAK** procedure was called.

A programmer may desire that the procedure not notify the user if an error code is returned and this logic is easily deleted from the **SPEAK** procedure. It is basically incorporated as an aid in development when first installing the speech-to-text software and hardware to track down errors that might occur.

PROGRAM LISTING

```
program say;

{This program will say the text entered as parameters on
the command line. Input is limited only to 127 total
characters (due to limitation of Turbo Pascal). To use the
program, enter the command 'say' followed by the text you
wish spoken. Remember to end the text with a sentence
terminator, either a period(.), question mark(?), or an
exclamation point(!). Examples:

    say This is a mighty fine computer!

    say Do you want to delete all files?

    say It is now time to have a coffee break.

{$V-}                                   {compiler directive to
relax
                                        length of strings}

type    longstr = string[240];      {size of buffer for text
input}

        word    = string[80];       {size of buffer for a word
                                     input}

var     passage  :longstr;   {buffer for text that is input}
        param    :word;      {buffer for word that is input}
        numparam :integer;   {number of parameters (words)}
        i        :integer;   {an index for words}

{$I b:speak.inc}                        {interface procedure for
                                         speech}

begin
  fillchar(passage,240,' '); {clear the buffer}
  numparam:=paramcount;      {find number of words passed}
  for i:= 1 to numparam do   {create the text buffer}
    begin                    {to be spoken}
      param:=paramstr(i);    {get each word from the command
                              line}
      passage:=passage+' '+param;  {and add it to the text
                                    buffer}
    end;                     {end for i:=1 to numparam}
  i:=length(passage);        {find the text length}
  if not (passage[i] in ['.','?','!']) then
     passage:=passage+'.';   {default to period if not
                              punctuated}
  speak(passage,65,170);     {speak the text in buffer,
                              pitch 65, rate 170}
end.
```

REFERENCES:

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

3. *Text-to-Speech Interface for the IBM Voice Communications Adapter, Talbot, Summer 1987*

CHAPTER 3

PROGRAM SAYTEXT.COM

Program SAYTEXT.COM

User Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

SAYTEXT is a program written in Turbo Pascal, Version 3.0, that allows a user the option of having the text within a file to be be translated into spoken language. The user enters the program name, **SAYTEXT**, followed by a parameter giving the filename of the text file that contains text that is desired to be spoken. The text file can be composed of an unlimited number of sentences each having up to 240 regular ASCII characters. Each sentence must end with a period (.), a question mark (?), or an exclamation point (!).

As an example, the user would enter '**saytext words.txt**' to have the text in the file **words.txt** translated into spoken language over an attached speaker.

Text-to-speech translation is made possible by use of the **SPEAK** procedure which serves as an interface to the **IBM Voice Communications Operating System** which in turn drives the **IBM Voice Communications Adapter** that produces intelligible speech from text. A user must assure that both the **IBM Voice Communications Adapter** (hardware) and the **IBM Voice Communications Operating System** (software) are installed on the machine at which they are working for the **SAYTEXT** program to work correctly.

This guide is intended to simplify the task of a user wishing to produce speech from text contained within a given text file. The sections following will discuss hardware and software requirements that are necessary to use this program. Also, installation instructions for using the **SAYTEXT** program will be discussed. An overview of using the **SAYTEXT** command is covered under the operating instructions section and an example is provided. Next, input and output formats and descriptions are discussed. Finally, references for further investigation are provided.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware:

Minimum hardware requirements are given in Table 3.1.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-sided diskette drive (360 KB / 1.2 MB) and one fixed disk

(cont.)

- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector)

Table 3.1

Software:

Minimum software requirements are given in Table 3.2.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0 (for compilation purposes only)
- SAYTEXT.COM

Table 3.2

INSTALLATION INSTRUCTIONS

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **IBM Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette named *vcapi.* The **Voice Communications Operating Subsystem Program** diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the **autoexec.bat** file:

set vcapi = y:\vcapi

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

```
y:\vcapi\vcapidrv /o 10
```

(the /o 10 option allows the text-to-speech function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

OPERATING INSTRUCTIONS

The **SAYTEXT** program will produce intelligible speech from the text within the file whose name is entered as a parameter on the command line. Any number of sentences may be included in the file but each is limited to a maximum of 240 regular ASCII text characters and must end with either a period (.), a question mark (?), or an exclamation point (!). If a sentence of more than 240 characters is entered or if termination punctuation is omitted, then a default period (.) is added either as the 240th character or at the end of the sentence.

Sentence intonation varies according to the terminator provided so the user should stress adding the desired termination punctuation. Pitch and rate of speech are set at 65 and 170 words per minute within the **SAYTEXT** program. If the user desires to change these rates, they may change the parameters passed in the call to the **SPEAK** procedure within the program and recompile the program. When

recompiling, make sure that the option to produce a .COM file has been selected from within Turbo Pascal. For additional information on the **SPEAK** procedure and pitch and rate, see reference 3 given at the end of this guide.

An example use of the **SAYTEXT** program follows:

if the file named **HELLO.DAT** contains the following text:

Hello all! It is so nice of you to visit. Will you come in and stay for awhile?

then to have the passage spoken, enter the following at the DOS command line:

A>saytext hello.dat

Error codes may be returned directly by the **SAYTEXT** program to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 3.3. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

ERROR CODES

Command	Error Code	Explanation
Open	0	Successful
	2	API inoperative
	16	RCB not available
	64	Invalid card number
Claimhdw	0	Successful
	2	Card Inoperative
	4	RCB invalid or not open
	16	At least one resource seized
	32	Unsupported hardware
	48	Combination of 16 and 32 above
Conndtop	0	Successful
	2	Card inoperative
	4	RCB invalid or not open
	16	Port or devices not claimed
	32	Unsupported devices
	64	Unsupported connection
Connftop	0	Successful
	2	Card inoperative

(cont.)

	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency
	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped
Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided

(cont.)

	128	No null found in text
	256	Pause pending (must call resume first)
Close	0	Successful
	2	Card inoperative
	4	RCB invalid or not open

Table 3.3

If a return code other than 0 is returned when a command within the **SPEAK** procedure is executed by the driver, then a message is written to the console telling which command returned the error and which error code was returned. The only ones which a user might see are error codes 16 or 64 from the speak command. This usually indicates that the end of sentence punctuation was not provided when the SPEAK procedure was called.

A programmer may desire that the procedure not notify the user if an error code is returned and this logic is easily deleted from the **SPEAK** procedure. It is basically incorporated as an aid in development when first installing the speech-to-text software and hardware to track down errors that might occur.

INPUT AND OUTPUT FORMATS AND DESCRIPTIONS

Input to the **SAYTEXT** program is a text file consisting of any number of sentences composed of 240 or fewer ASCII text characters that end with a sentence terminator, either a period (.), a question mark (?), or an exclamation point (!).

Output is in the form of intelligible spoken translation of the text that is input. Other output is in the form of error messages directly to the user console.

REFERENCES:

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

3. *Text-to-Speech Interface for the IBM Voice Communications Adapter, Talbot, Summer 1987*

Program SAYTEXT.COM

System Documentation Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

SAYTEXT is a program written in Turbo Pascal, Version 3.0, that allows a user to have text in a file translated into intelligible spoken language. The program provides a quick and easy way of producing speech from a text file entered at the DOS command line as a parameter to the program.

For example, to have the text in the file **text.fil** translated into voice, the user would enter "**saytext text.fil**" at the DOS command prompt. This command will activate the **SAYTEXT** program and cause the text within **text.fil** to be voiced over a speaker. A user must assure that both the **IBM Voice Communications Adapter** (hardware) and the **IBM Voice Communications Operating System** (software) are installed on the machine at which they are working for the **SAYTEXT** program to work correctly.

Text-to-speech translation is made possible by use of the **SPEAK** procedure which serves as an interface to the **IBM Voice Communications Operating System** which in turn drives the **IBM Voice Communications Adapter** that produces intelligible speech from text.

SYSTEM SPECIFICATIONS

Both the **IBM Voice Communications Operating System** and the **IBM Voice Communications Application Program Interface** software are required for operation of the **SAYTEXT.COM** program. Details on installation are provided below.

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **IBM Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette named *vcapi*. The **Voice Communications Operating Subsystem Program** diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the **autoexec.bat** file:

set vcapi = y:\vcapi

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing he API code.)

y:\vcapi\vcapidrv /o 10

(the /o 10 option allows the text-to-speech function to to be loaded when the API driver, vcapidrv, is loaded at boot time.)

HARDWARE REQUIREMENTS

Minimum hardware requirements are given in Table 3.4.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-sided diskette drive (360 KB / 1.2 MB) and one fixed disk

(cont.)

- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector)

Table 3.4

SOFTWARE REQUIREMENTS

Minimum software requirements are given in Table 3.5.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0 (for compilation purposes only)
- SAYTEXT.COM

Table 3.5

DESIGN DETAILS

The **SAYTEXT.COM** program was developed to allow a user to produce speech from any ASCII text file. The filename of a file which contains an unlimited number of complete sentences is passed to the program as a parameter on the command line. The file is then read in sentence-by-sentence and stored in a linked list. Next, each sentence is passed to the **SPEAK** procedure, which serves as an interface to the **IBM Voice Communications Adapter** which translates text into speech.

IMPLEMENTATION DETAILS

The **SAYTEXT.COM** program uses the **SPEAK** procedure (see reference 3) to interface the **IBM Voice Communications Interface Program** to issue commands to the **IBM Voice Communications Adapter**. The filename of a file which contains sentences composed of up to 240 standard ASCII text characters is passed as a parameter on the command line. The text file can have an unlimited number of sentences and each sentence must end with a period (.), a question mark (?), or an exclamation point (!). If an invalid filename is passed as the parameter, then the program aborts and notifies the user that the filename

does not exist. For valid filenames, the file is read in one sentence at a time into a linked list to reduce disk access time. Once the entire file is read in, then each sentence is passed to the SPEAK procedure to be spoken.

Pitch and rate of speech are set to 65 and 170 words per minute within the **SAYTEXT** program. If the user desires to change these rates, they may change the parameters passed in the call to the **SPEAK** procedure within the program and recompile the program. When recompiling, make sure that the option to produce a .COM file has been selected from within Turbo Pascal. For additional information on the **SPEAK** procedure and pitch and rate, see the reference 3 given at the end of this guide.

An example use of the **SAYTEXT** program follows:

if the file named **HELLO.DAT** contains the following text:

Hello all! It is so nice of you to visit. Will you come in and stay for awhile?

then to have the passage spoken, enter the following at the DOS command line:

A>saytext hello.dat

Error codes may be returned directly by the **SAYTEXT** program to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 3.6. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

ERROR CODES

Command	Error Code	Explanation
Open	0	Successful
	2	API inoperative
	16	RCB not available
	64	Invalid card number
Claimhdw	0	Successful
	2	Card Inoperative
	4	RCB invalid or not open
	16	At least one resource seized
	32	Unsupported hardware
	48	Combination of 16 and 32 above

(cont.)

Conndtop	0	Successful
	2	Card inoperative
	4	RCB invalid or not open
	16	Port or devices not claimed
	32	Unsupported devices
	64	Unsupported connection
Connftop	0	Successful
	2	Card inoperative
	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency
	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped

(cont.)

Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided
	128	No null found in text
	256	Pause pending (must call resume first)
Close	0	Successful
	2	Card inoperative
	4	RCB invalid or not open

Table 3.6

If a return code other than 0 is returned when a command within the **SPEAK** procedure is executed by the driver, then a message is written to the console telling which command returned the error and which error code was returned. The only ones which a user might see are error codes 16 or 64 from the speak command. This usually indicates that the end of sentence punctuation was not provided when the **SPEAK** procedure was called.

A programmer may desire that the procedure not notify the user if an error code is returned and this logic is easily deleted from the **SPEAK** procedure. It is basically incorporated as an aid in development when first installing the speech-to-text software and hardware to track down errors that might occur.

PROGRAM LISTING

```
{************************************************************
The SAYTEXT program causes the text in the file, whose name
is passed as a parameter, to be spoken.  The file should be
in regular ASCII characters similiar to this passage
following all rules of normal punctuation.  The length of
the input file is unlimited.  The text is first read into a
linked list then each node of the linked list is spoken.

Example:

if the file named HELLO.DAT contains the following text:

      Hello all!  It is so nice of you to visit.  Will you
      come in and stay for awhile?

then to have the passage spoken, enter the following
command:

      saytext hello.dat
************************************************************}
program saytext;

{$V-}  {compiler directive to relax length of parameter
        strings passed}

type   longstr = string[240];     {length of text string to
                                   speak}
       filname = string[66];      {file name passed}
       buffer  = array[1..240] of char; {temporary buffer
                                         storage}
```

```
{******* RECORD FOR LINKED LIST NODE ***********}
    LlistNod = ^SNode;
    SNode = record
            txt: longstr;
            next: LlistNod;
            prior: LlistNod;
    end;

{******* RECORD FOR LINKED LIST HEADER ***********}
    Slist = ^SHead;
    SHead = record
            length: integer;
            first:  LlistNod;
            Last:   LlistNod;
    end;

var     data      :filname;       {buffer to hold file name
                                    that is passed}
        datafile  :text;          {assigned to the filename}
        i,j       :integer;       {counter for nodes and
                                    chars}
        LList     :Slist;         {the head node}
        node      :LListnod;      {pointer to keep track of
                                    current node}
        Str240    :longstr;       {buffer for string}
        chin      :char;          {char read in}
        buf       :buffer;        {used to manipulate data}

{*********************************************************}
{*     function to test for existence of a file          *}
{*********************************************************}
Function Exist(filename: filname): boolean;

var     fil       :file;

begin
  assign(fil,filename);
  {$I-}
  reset(fil);
  {$I+}
  exist:= (IOresult = 0)
end;{function exist}

{*********************************************************
*Node_Ptr; RETURNS A PTR  TO  CURRENT NODE OF LINKED LIST
*********************************************************}

Function Node_Ptr(pos: integer): LlistNod;

Var
   i:  integer;
   nd: LlistNod;
```

```
Begin
  nd := Llist^.first;
  for i := 2 to pos do
    nd := nd^.next;
  Node_Ptr := nd;
End;


{************************************************************
* CreateLst; CREATES HEADER FOR LINKED LIST FOR TEXT LINES*
************************************************************}

Function CreateLst: Slist;

Var
   thishead: Slist;

Begin
  new(thishead);
  thishead^.length := 0;
  thishead^.first := nil;
  thishead^.last := nil;
  CreateLst := thishead;
End;

{************************************************************
* Make_Node;CREATES NEW NODE FOR LINKED LIST                *
************************************************************}

Function  Make_Node(dat:  longstr;  prev,  nxt:  LlistNod):
LlistNod;

Var
   thisone: LlistNod;

Begin
  new(thisone);
  thisone^.txt := Copy(dat,1,Length(dat));
  thisone^.prior := prev;
  thisone^.next := nxt;
  Make_Node := thisone;
End;

{************************************************************
* APP_Llist; APPENDS A NODE ONTO LINKED LIST                *
************************************************************}
Procedure App_Llist(dat: longstr);

Var
   thisone: LlistNod;
```

```
Begin
  if Llist^.first = nil then
    begin
      thisone := Make_Node(dat,nil,nil);
      Llist^.last := thisone;
      Llist^.first := thisone;
    end
  else
    begin
      thisone := Make_Node(dat,Llist^.last,nil);
      Llist^.last^.next := thisone;
      Llist^.last := thisone;
    end;
  Llist^.length := Llist^.length + 1;
End;

{************************************************************
*  DelHere;  DELETES A NODE FROM THE TEXT LINKED LIST AND *
*        RETURNS THE TEXT STRING FROM THAT NODE           *
************************************************************}
Function DelHere(pos: integer): longstr;

Var
   temp: LlistNod;

Begin
  temp := Llist^.first;
  if pos = 1 then
    begin
      Llist^.first := temp^.next;
      if Llist^.first <> nil then
        Llist^.first^.prior := nil;
    end
  else
    begin
      temp := Node_Ptr(pos);
      temp^.prior^.next := temp^.next;
      if temp^.next = nil then
        Llist^.last := temp^.prior
      else
        temp^.next^.prior := temp^.prior;
    end;
  DelHere := temp^.txt;
  Dispose(temp);
  Llist^.length := Llist^.length - 1;
End;
```

```
{*************************************************************
*   DEALL_LIST;                                              *
*************************************************************}
Procedure Deall_List;

Var
  Tx : String[80];

Begin
  while Llist^.length > 0 do
    Tx := DelHere(1);
  Dispose(Llist);
End;

{$I b:speak.inc}                    {speech interface procedure}

begin
  data:=paramstr(1);                {get the file name passed as
                                     a parameter}
  if exist(data) then               {see if the filename is
                                     valid}
    begin                           {do this if filename valid
                                     else tell user}
      assign(datafile,data);        {assign var datafile to the
                                     string name}
      reset(datafile);              {get the file ready to read}
      LList:=CreateLst;             {create a head node}
      while not eof(datafile) do
        begin                       {begin while not eof...}
          j:=1;                     {initialize char counter}
          repeat
            read(datafile,chin);        {read char in}
            buf[j]:=chin;               {put it in an array}
            j:=j+1;                     {increment the index}
          until (chin in ['.','?','!']) or (j > 240) or
                eof(datafile);
                                    {stop for end of sentence or
                                     buffer full or end of file}
          Str240:=copy(buf,1,j-1);  {creates a complete
                                     sentence}
          APP_LList(Str240);        {add it to the array}
        end;                        {while not eof(datafile)}
      close(datafile);              {remember to close the file}
      node:=LList^.first;           {set pointer to first node}
      for i:=1 to LList^.length do
        begin                       {for i:=1 to LList...}
          speak(node^.txt,65,175); {speak the current line}
          node:=node^.next;         {move the pointer up}
        end;                        {for i:=1 to LList...}
      Deall_List;                       {delete all the nodes}
    end                             {while not eof...}
```

```
    else
      writeln(data,' does not exist.');
                                {error message if file does not
                                 exist}
end.                            {program SAYTEXT}
```

REFERENCES

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

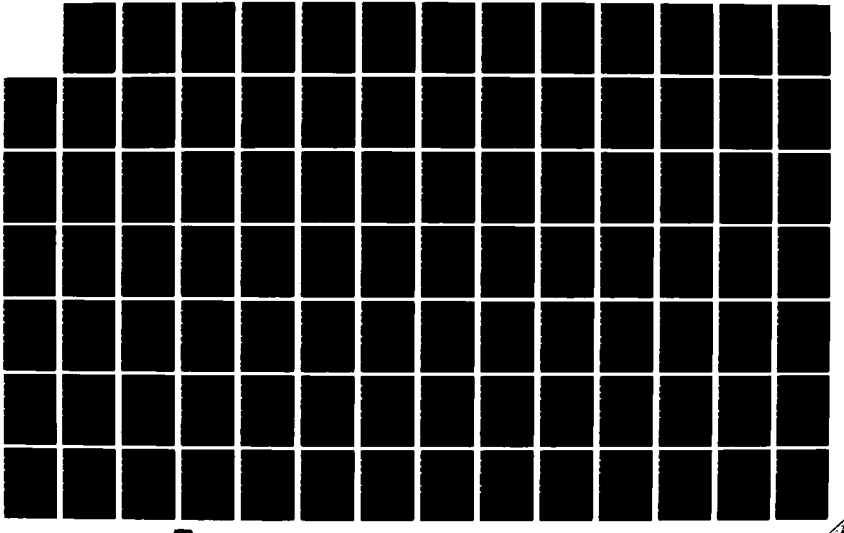
3. *Text-to-Speech Interface for the IBM Voice Communications Adapter, Talbot, Summer 1987*

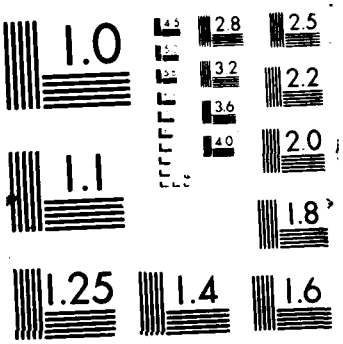
CHAPTER 4

PROGRAM REMIND.COM

AD-A186 743 VOICE TECHNOLOGY USING PERSONAL COMPUTERS(U) AIR FORCE 2/3
INST OF TECH WRIGHT-PATTERSON AFB OH G L TALBOT 1987
AFIT/CI/NR-87-43T

UNCLASSIFIED F/G 25/4 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

Program REMIND.COM

User Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

REMIND is a program written in Turbo Pascal, Version 3.0, that provides the user with the ability to have prepared messages spoken at desired times. When the program is first started, the user is prompted for messages and the time when each message should be spoken. Once the user's desired schedule is correct, the program becomes memory resident and any other normal activity may be resumed. At the designated times, each message will be spoken.

The user can review/revise the schedule or terminate the REMIND program at any time by entering the keystroke, **ALT/F7**. Messages are limited to 127 regular ASCII text characters and must end with termination punctuation, either a period (.), a question mark (?), or an exclamation point (!). Times are input using a twenty-four hour clock in the format hh:mm. For example, ten o'clock a.m. is entered as 10:00 while three thirty-four p.m. is entered as 15:34.

Text-to-speech translation is made possible by use of the **SPEAK** procedure which serves as an interface to the **IBM Voice Communications Operating System** which in turn drives the **IBM Voice Communications Adapter** that produces intelligible speech from text. A user must assure that

both the **IBM Voice Communications Adapter** (hardware) and the **IBM Voice Communications Operating System** (software) are installed on the machine at which they are working for the **REMIND** program to work correctly.

This guide is intended to simplify the task of a user wishing to produce speech from text at designated times. The sections following will discuss hardware and software requirements that are necessary to use this program. Also, installation instructions for using **REMIND** will be discussed. An overview of using the **REMIND** program is covered under the operating instructions section and an example is provided. Next, input and output formats and descriptions are discussed. Finally, references for further investigation are provided.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware:

Minimum hardware requirements are given in Table 4.1.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-sided diskette drive (360 KB / 1.2 MB) and one fixed disk
- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector)

Table 4.1

Software:

Minimum software requirements are given in Table 4.2.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0 (for compilation purposes only)
- REMIND.COM

Table 4.2

INSTALLATION INSTRUCTIONS

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **IBM Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette named *vcapi*. The Voice

Communications Operating Subsystem Program diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the **autoexec.bat** file:

set vcapi = y:\vcapi

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

y:\vcapi\vcapidrv /o 10

(the /o 10 option allows the text-to-speech function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

OPERATING INSTRUCTIONS

To activate the **REMIND** program, enter **remind** at the DOS command line then follow the instructions provided. Up to fifteen separate messages and designated times may be

input. If less than the fifteen messages are desired, then the user may enter 'q' or 'Q' to quit at any time. When all desired messages have been entered, the schedule will be displayed for the user to review and revise as desired. Once correct, the program and schedule become memory resident and control returns to the DOS command line. The program begins checking the time every 15 seconds and compares this time to the times within the schedule. If the times match, then the associated message is voiced through an attached speaker. The user may enter an **ALT/F7** at any time, even from within other programs, to review/revise the schedule or terminate the **REMIND** program.

An example session using the **REMIND** program follows. User input is italicized.

The user enters **remind** to activate the program in Figure 4.1.

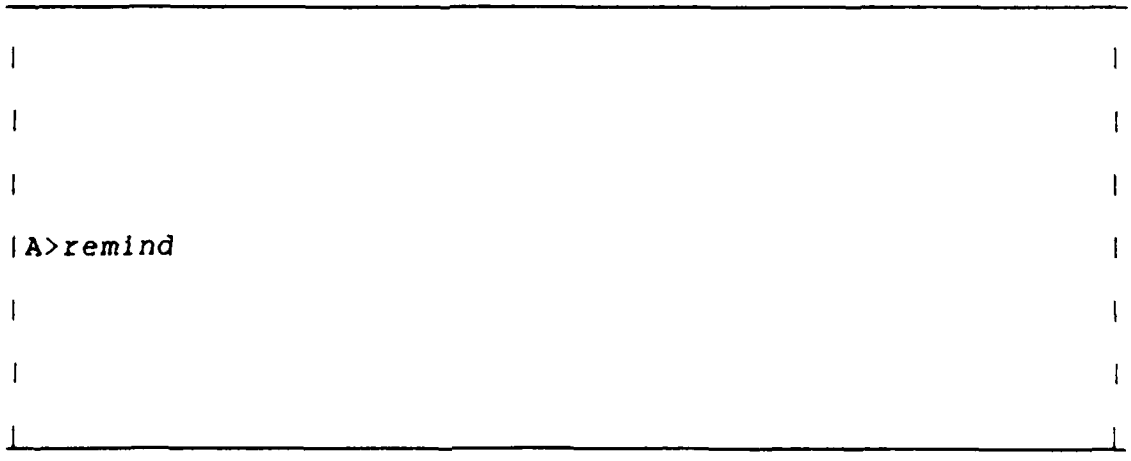


Figure 4.1

The information in Figure 4.2 appears in a window to the screen.

```
|                                                              |
| Enter the time (hh:mm) then the message you wish spoken.     |
| Time range is 00:00-23:59.  Message is a maximum of 127      |
| characters.  15 different messages may be entered.           |
|                                                              |
|   Time 1?  (hh:mm)              Q to quit.                   |
|   10:30                                                      |
|   Message 1?       Punctuation required.                     |
|   Time is ten-thirty.  It's coffee break time.               |
|                                                              |
|                                                              |
|______________________________________________________________|
```

Figure 4.2

The screen for second and succeeding entries appears in Figure 4.3.

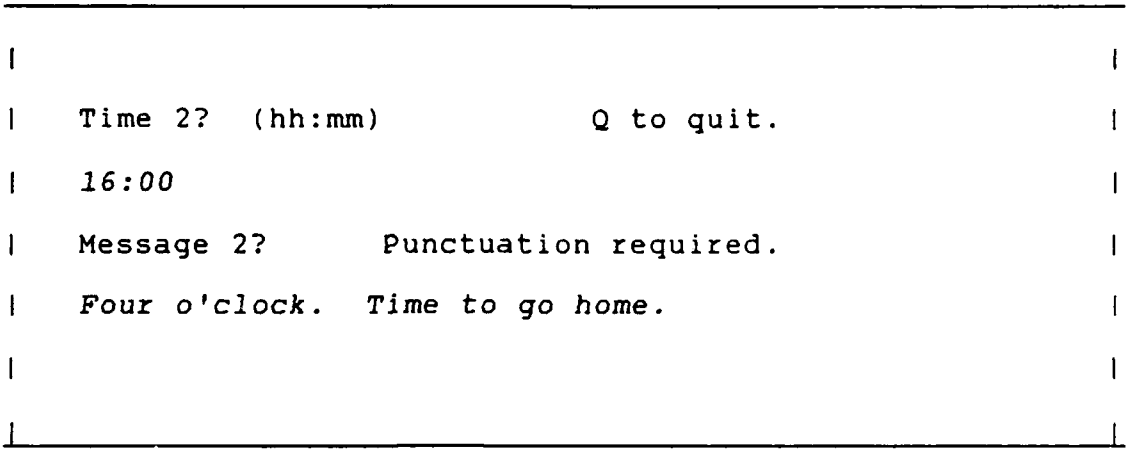


Figure 4.3

Figure 4.4 demonstrates when the user has completed inputting entries.

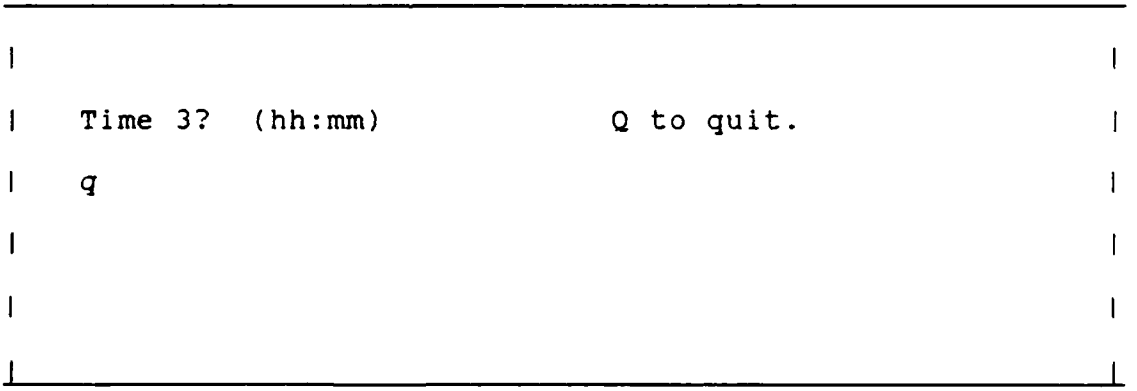


Figure 4.4

Next, in Figure 4.5, the user is allowed to review information that has been input.

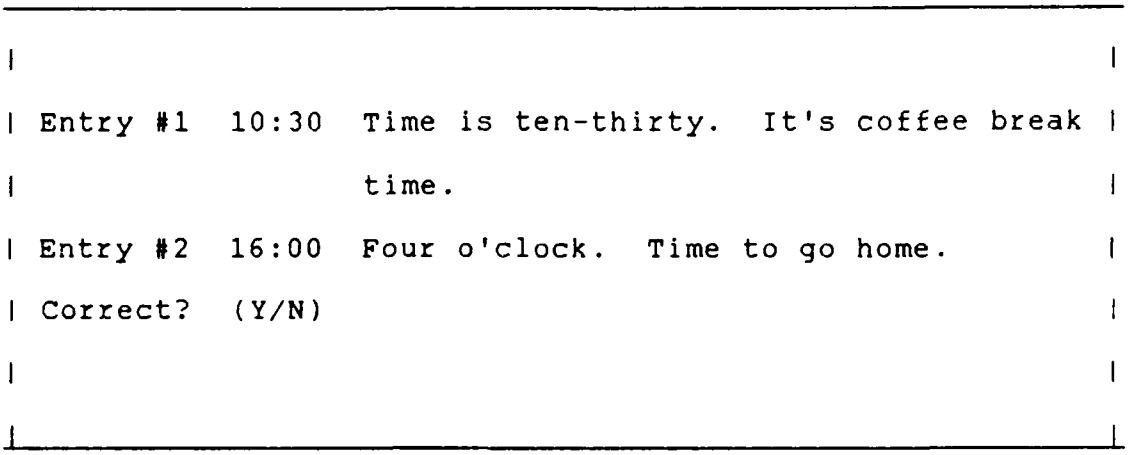


Figure 4.5

If the user enters 'N' or 'n', then the screen in Figure 4.6 is displayed, if 'Y' or 'y', then Figure 4.7 is shown.

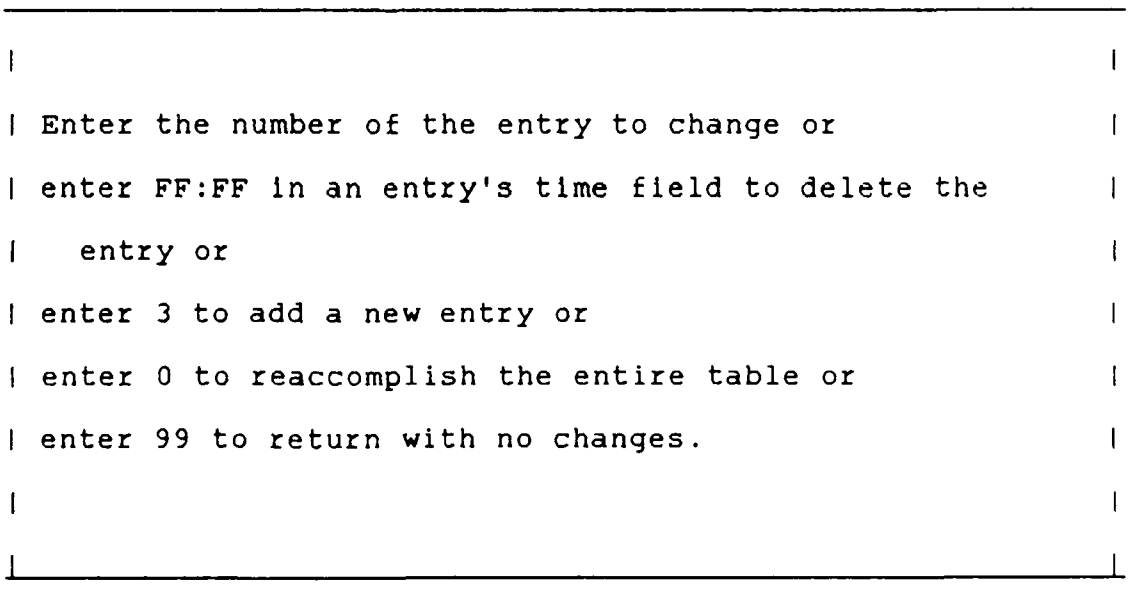


Figure 4.6

Once the schedule is correct, the user enter 'Y'or 'y' and the program becomes memory resident and the user sees the screen in Figure 4.7 before the DOS command line is returned.

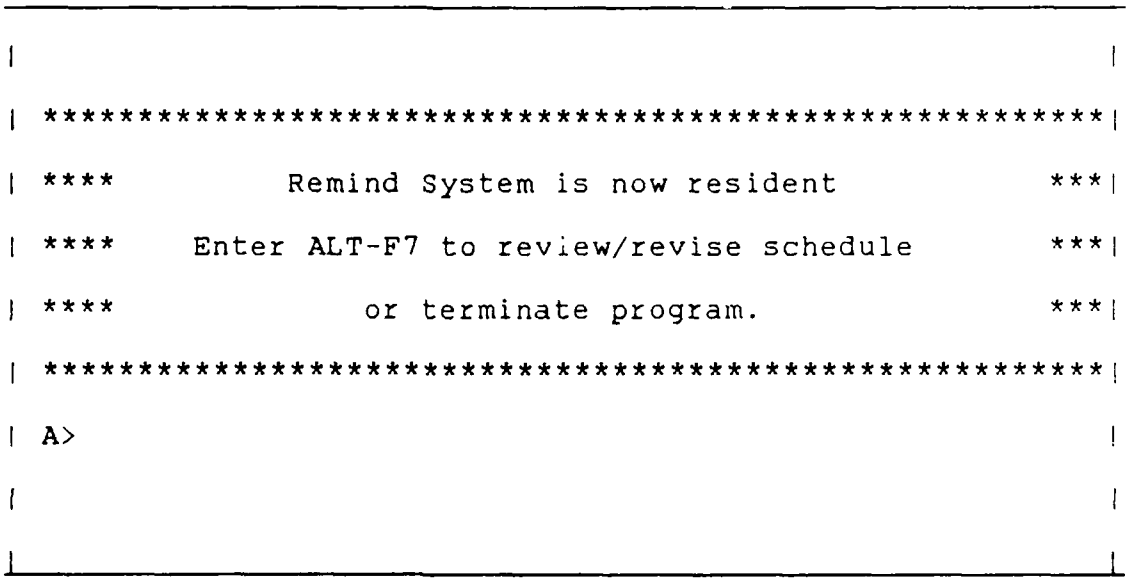


Figure 4.7

If at any time thereafter, the user enters **ALT/F7**, the screen in Figure 4.8 is displayed.

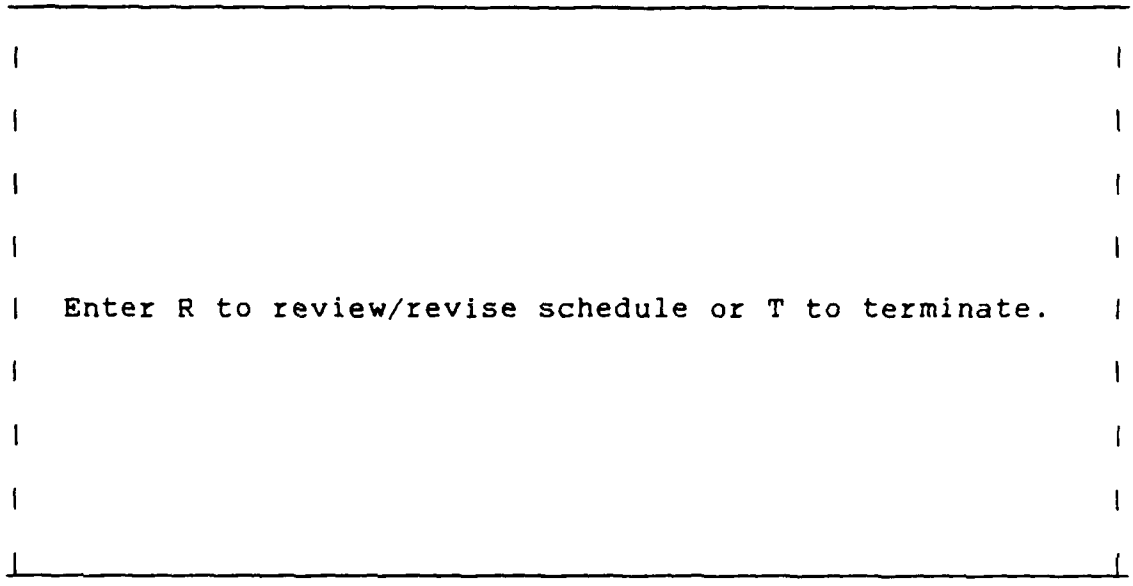


Figure 4.8

Error codes may be returned directly by the **REMIND** program to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 4.3. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency
	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped
Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided

(cont.)

REFERENCES

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

3. *Text-to-Speech Interface for the IBM Voice Communications Adapter, Talbot, Summer 1987*

Program REMIND.COM

System Documentation Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

Communications Operating Subsystem Program diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required speech-to-text function, the following commands should be placed in the **autoexec.bat** file:

```
set vcapi = y:\vcapi\
```

(where y is the drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

```
y:\vcapi\vcapidrv /o 10
```

(the /o 10 option allows the text-to-speech function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

HARDWARE REQUIREMENTS

Minimum hardware requirements are given in Table 4.4.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 256 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-sided diskette drive (360 KB / 1.2 MB) and one fixed disk
- Monochrome or color monitor
- An IBM Voice Communications Adapter
- Speaker (8-ohm, capable of handling two watts of audio power with an attached subminiature 2.5 mm (0.1 inch) connector)

Table 4.4

SOFTWARE REQUIREMENTS

Minimum software requirements are given in Table 4.5.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- Turbo Pascal, Version 3.0 (for compilation purposes only)
- REMIND.COM

Table 4.5

DESIGN DETAILS

The **REMIND.COM** program uses a memory resident shell and the **SPEAK** procedure (see reference 3) to interface the **IBM Voice Communications Interface Program** to issue commands to the **IBM Voice Communications Adapter**. A message that is entered to be spoken at a specific time is spoken when the internal machine time matches the desired time for speech.

IMPLEMENTATION DETAILS

To activate the **REMIND** program, enter **remind** at the DOS command line then follow the instructions provided. Up to fifteen separate messages and designated times may be input. If less than the fifteen messages are desired, then the user may enter 'q' or 'Q' to quit at any time. When all desired messages have been entered, the schedule will be displayed for the user to review and revise as desired. Once correct, the program and schedule become memory resident and control returns to the DOS command line. The program begins checking the time every 15 seconds and compares this time to the times within the schedule. If the times match, then the associated message is voiced through an attached speaker.

The user may enter an **ALT/F7** at any time, even within other programs, to review/revise the schedule or terminate the **REMIND** program.

Several variables may be changed within the program. First, the time that the program goes out to check the current time is set to 15 seconds. This may be changed by changing the variable **TIMER__TIME** to the desired time in seconds. Next, the hot key, **ALT/F7**, may be changed by changing the constant **Our_HotKey** to the desired scan code.

Third, the constant **MaxMsg** may be set to the maximum number of messages desired. It is currently set to 15. The program is written in Turbo Pascal, Version 3.0. The memory resident shell was taken in part from *The Hunter's Helper*, by Lane Ferris, et. al. When resident, the program takes approximately 34K of memory.

The main program, **REMIND.PAS**, requires several include files for compilation. These are given in Table 4.6.

FILES REQUIRED FOR REMIND.PAS COMPILATION

- **SPEAK.INC:**	Interface to **IBM Voice Communications Adapter**
- **STAYWNDO.341:**	Used to create popup windows
- **STAYXIT.420:**	Used in program termination
- **STAYSUBS.420:**	Common subroutines
- **STAYI16.410:**	Handles interrupt 16 calls
- **STAYI13.410:**	Handles interrupt 13 calls
- **STAYI21.410:**	Handles interrupt 21 calls
- **STAYI8.420:**	Handles interrupt 8 calls
- **STAYI28.410:**	Handles interrupt 28 calls
- **STAYSAVE.420:**	Saves OS system structures
- **STAYRSTR.420:**	Used to terminate and remain resident
- **CLKI8.410:**	Provides the timer

Table 4.6

The program should be compiled as a COM file with minimum dynamic memory set to 100 and maximum dynamic memory set to 200. This ensures enough memory is allocated to the heap to avoid collisions while limiting the amount of memory reserved for the heap.

An example session using the **REMIND** program follows. User input is italicized.

The user enters **remind** to activate the program in Figure 4.9.

```
A>remind
```

Figure 4.9

The information in Figure 4.10 appears in a window to the screen.

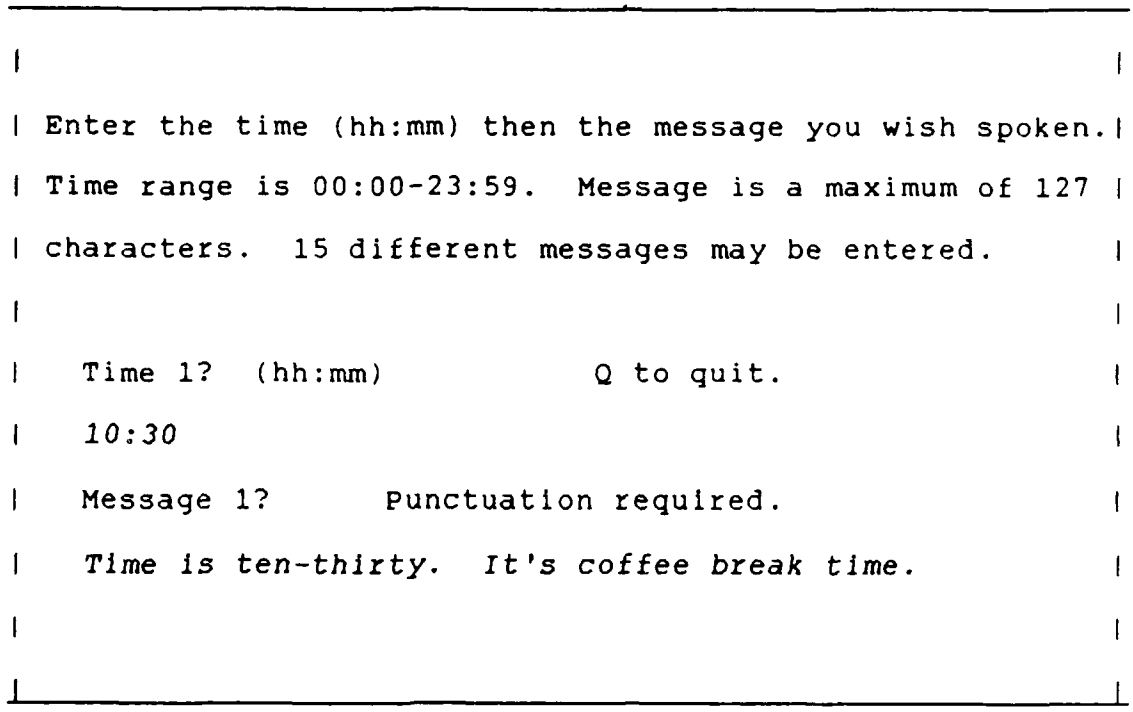


Figure 4.10

The screen for second and succeeding entries appears in Figure 4.11.

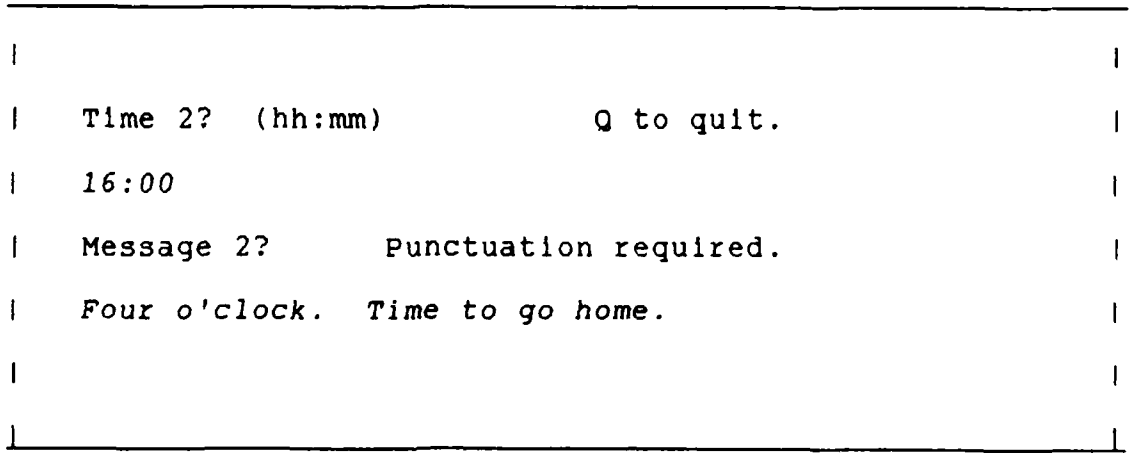


Figure 4.11

Figure 4.12 demonstrates when the user has completed inputting entries.

```
|                                                          |
|   Time 3?  (hh:mm)              Q to quit.               |
|   q                                                      |
|                                                          |
|                                                          |
|__________________________________________________________|
```

Figure 4.12

Next, in Figure 4.13, the user is allowed to review information that has been input.

```
|                                                          |
| Entry #1  10:30  Time is ten-thirty.  It's coffee break  |
|                  time.                                   |
| Entry #2  16:00  Four o'clock.  Time to go home.         |
| Correct?  (Y/N)                                          |
|                                                          |
|__________________________________________________________|
```

Figure 4.13

If the user enters 'N' or 'n', then the screen in Figure 4.14 is displayed, if 'Y' or 'y', then Figure 4.15 is shown.

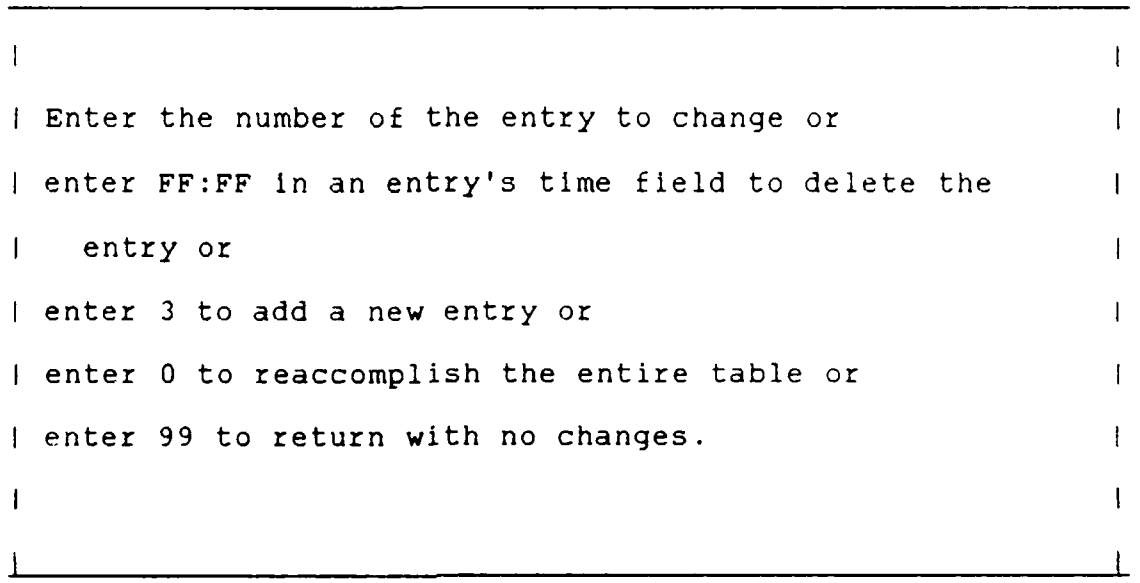


Figure 4.14

Once the schedule is correct, the user enter 'Y'or 'y' and the program becomes memory resident and the user sees the screen in Figure 4.15 before the DOS command line is returned.

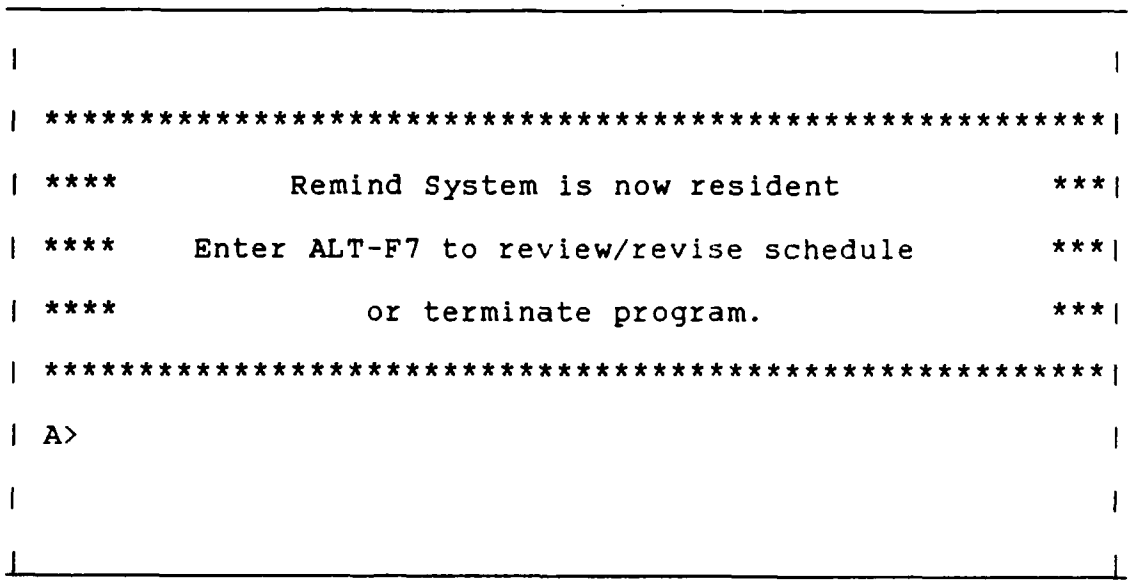


Figure 4.15

If at any time thereafter, the user enters **ALT/F7**, the screen in Figure 4.16 is displayed.

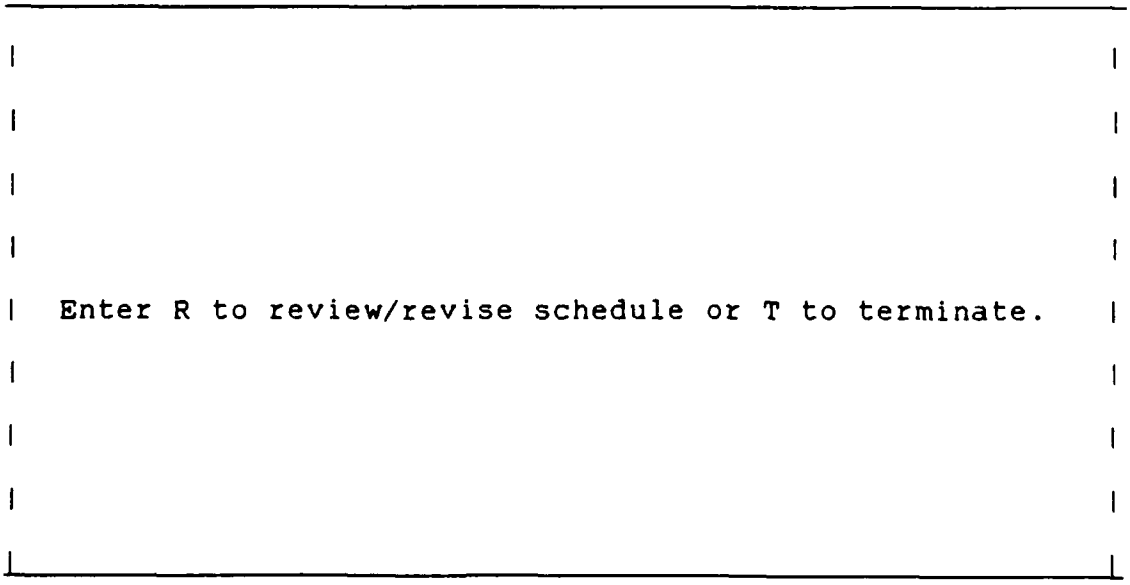


Figure 4.16

Error codes may be returned directly by the **REMIND** program to the user console whenever an abnormal return code (one other than 0) is returned from the Voice Driver. A quick synopsis of these error codes is given in Table 4.7. For a more complete description, consult the *IBM Voice Communications Application Program Interface Reference, Vol 1.*

ERROR CODES

Command	Error Code	Explanation
Open	0	Successful
	2	API inoperative
	16	RCB not available
	64	Invalid card number
Claimhdw	0	Successful
	2	Card Inoperative
	4	RCB invalid or not open
	16	At least one resource seized
	32	Unsupported hardware
	48	Combination of 16 and 32 above

(cont.)

Conndtop	0	Successful
	2	Card inoperative
	4	RCB invalid or not open
	16	Port or devices not claimed
	32	Unsupported devices
	64	Unsupported connection
Connftop	0	Successful
	2	Card inoperative
	4	CID invalid
	16	Port or partition not claimed
	32	Function set not known
	64	Function set not accessible
	128	Insufficient storage
	256	Port not specified
	512	Function set already connected to CID
	1024	Unsupported concurrency
	2048	Function set cannot be held by partition
	4096	Invalid configuration
Initialize	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Function set not stopped

(cont.)

Speak	0	Successful
	4	Function set not connected
	8	Busy (re-entrant call)
	16	Syntax error
	32	Pause command received
	64	Input buffered since no sentence terminator provided
	128	No null found in text
	256	Pause pending (must call resume first)
Close	0	Successful
	2	Card inoperative
	4	RCB invalid or not open

Table 4.7

If a return code other than 0 is returned when a command within the **SPEAK** procedure is executed by the driver, then a message is written to the console telling which command returned the error and which error code was returned. The only ones which a user might see are error codes 16 or 64 from the speak command. This usually indicates that the end of sentence punctuation was not provided when the **SPEAK** procedure was called.

A programmer may desire that the procedure not notify the user if an error code is returned and this logic is easily deleted from the **SPEAK** procedure. It is basically incorporated as an aid in development when first installing the speech-to-text software and hardware to track down errors that might occur.

PROGRAM LISTING

```
{$R+}
{$C-}
{$V-}

PROGRAM REMIND;
{This program is a memory resident program that drives the
IBM voice applications software and hardware board. When
first loaded, it allows input of messages to be spoken and
the time when they should be spoken. Interface to the board
is made through the procedure speak. After the user is
prompted for input (messages and times), the program
terminates and becomes memory resident. Access to the
schedule for review/revision or to terminate the program
can be made by entering ALT-F7.}


{ * * * * * CONSTANTS * * * * * * * * * * * * * * * * * * * * }
  const
{the next field is needed for the windo.inc routines }
    MaxMsg          = 15;  {maximum number of messages to
                              be in schedule}
    MaxWin          = 10;  {Max number of windows open at
                              one time }
    Esc             = #27; {character equivalent of Escape
                              Key}
    Alt             = 08;  {Shift bits at 40:17 }
    Ctrl            = 04;
    Left_Shift      = 02;
    Rght_Shift      = 01;
```

```
    BIOSI8          = 8;    {Bios Timer interrupt}
    BIOSI16         = $16; {Bios Keyboard interrupt}
    BIOSI13         = $13; {Bios Disk interrupt}
    DOSI21          = $21; {DOS service router interrupt}
    DOSI28          = $28; {DOS Idle interrupt}
{------------ T Y P E   D E C L A R A T I O N S -------}
  Type
    Regtype     = record
                   Ax,Bx,Cx,Dx,Bp,Si,Di,Ds,Es,Flags:integer
                  end;

    HalfRegtype = record
                   Al,Ah,Bl,Bh,Cl,Ch,Dl,Dh:byte
                  end;

    filename_type = string[64];
    Vector    = record      { Interrupt Vector type   }
                  IP,CS :integer ;
                end ;
    longstr = string[24];
{-------------- T Y P E D   C O N S T A N T S ----------}
  Const

    Our_HotKey    : byte = 110;    { scan code for ALT-F7}

{****** scan code can be changed to make *****************}
{****** another key active as the hot key. ***************}

      { This table marks those INT 21 functions which must
be passed without modification. They either never return,
fetch parameters from the stack, or may be interrupted by a
TSR }

      Functab     : array[0..$6F] of byte =
              (1,1,1,1, 1,1,1,1, 1,1,1,1, 1,0,0,0,  {0-C}
               0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
               0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,0,1,  {26,2F}
               0,1,1,1, 1,1,0,0, 0,0,0,0, 0,0,0,0,  {31-35}
               0,0,0,0, 0,0,0,0, 1,1,1,1, 1,1,0,0,  {48-4D}
               1,1,1,1, 0,1,0,0, 1,0,0,0, 0,1,1,1,  {50-53,
                                                  ,55,58,5D-5F}
               1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1); {60-62}
Intr_Flags : byte = 0;      {Active interrupts flags}
INT13_on   =  04;          {Disk  interrupt is active}
INT21_on   =  08;          {DOS Service router is active}
Status      : byte = 0;     {Status of current TSR activity}
Hotkey_on  =  01;          {Received the HotKey}
Inuse      =  02;          {TSR is active}
Foxs       = $FF;          {workaround for inline hex FF}
DosVersion : byte = 0;      {Current Version of DOS}
WaitCount  : byte = 0;      {Wait to activate count}
```

```
UserProgram :integer = 0;  {Offset to Users Program Code}
OurDSeg: integer = 0;      {Turbo Data Segment Value  }
OurSSeg: integer = 0;      {Turbo Stack Segment Value   }
DosDSeg: integer = 0;      {Dos Datasegment value     }
DosSSeg: integer = 0;      {Dos Stack Segment Value   }
DosSPtr: integer = 0;      {Dos Stack pointer value   }
DosSsiz: integer = 0;      {Dos Stack size in words }
UsrDSeg: integer = 0;      {Interrupted Datasegment Value}
UsrSSeg: integer = 0;      {Interrupted Stack Segment
                              Value}
UsrSPtr: integer = 0;      {Interrupted Stack pointer
                              Value   }
OurPSP : integer = 0;

 { The following  constants *MUST* remain in the IP:CS
   order.  StaySave uses them as  JMP targets}

BIOS_INT8   : vector = (IP:0;CS:0);  {BIOS Timer Interrupt
                                        Vector }
BIOS_INT16  : vector = (IP:0;CS:0);  {BIOS Keyboard
                                        Interrupt Vector }
BIOS_INT13  : vector = (IP:0;CS:0);  {BIOS Disk Interrupt
                                        Vector    }
DOS_INT21   : vector = (IP:0;CS:0);  {DOS Sevice Interrupt
                                        Vector}
DOS_INT28   : vector = (IP:0;CS:0);  {DOS idle Service
                                        interrupt Vector}
DOSStat1    : vector = (IP:0;CS:0);  {Pointer to INDOS
                                        byte}
DOSStat2    : vector = (IP:0;CS:0);  {Pointer to CRITICAL
                                        byte}
Version  :string[4] = '4.15';   { Current Version number }
      {NEEDED FOR SETTIME}
TIMER_HI: INTEGER = 0;              {used to set timer}
TIMER_LO: INTEGER = 0;              {used to set timer}
TIMER_ON = 4;                       {timer mask bit}
FROM_TIMER = 8;                     {timer mask bit}
TIMER_TIME = 15;                    {check every 15 seconds}
{* CHANGE TIMER_TIME TO THE VALUE (IN SECONDS) TO THE*****}
{** YOU WANT THE PROGRAM TO CHECK FOR TIME EXPIRATION ****}

{-------------- V A R I A B L E S ----------------------}
    Var

      Regs       : regtype;
      HalfRegs   : halfregtype absolute regs;
      Keychr     : char ;
      Bytecount  : integer;
      SavedPSP   : integer;      { Program Segment Prefix
                                   pointers }
      Error      : integer;      { I/O results }
```

```
    Good      : boolean;        { I/O results switch }
    Terminate : boolean;        { Exit stayRes Flag  }

   OurDTA   :Array [1..2] Of integer; {Local DTA pointer}
   SavedDTA :Array [1..2] of integer; {Interrupted DTA
                                       pointer}
    {NEEDED FOR REMIND PROGRAM}
     HICLOCK: INTEGER ABSOLUTE $40 : $6E;
     LOCLOCK: INTEGER ABSOLUTE $40 : $6C;
     TICS                          : REAL;
     times  :array[1..MaxMsg] of string[5];
     mesg   :array[1..MaxMsg] of string[127];
     said   :array[1..MaxMsg] of boolean;
     line   :string[127];
     i,j,k  :integer;
     chin   :char;
     punct  :char;
{-------------------------------------------------------}
{   W  I  N  D  O  W     R  O  U  T  I  N  E            }
{-------------------------------------------------------}
{$I b:STAYWNDO.341}
{-------------------------------------------------------}
{   S  T  A  Y  E  X  I  T                              }
{-------------------------------------------------------}
{$I b:STAYXIT.420}
{*******************************************************}
{-------------------------------------------------------}
{THE FOLLOWING ARE THE USER INCLUDE ROUTINES             }
{-------------------------------------------------------}
{*******************************************************}
{$I b:STAYSUBS.420}
{-------------------------------------------------------}
{          PROCEDURE SETTIME NEEDED TO INITIALIZE       }
{-------------------------------------------------------}

{-------------------------------------------------------}
{         D o u b l e   to   R e a l   number conversion }
{-------------------------------------------------------}
  function double_to_real(I,J : integer):real;
  var temp : real;
  begin
    temp := I; IF temp < 0 THEN temp := temp + 65536.0;
    temp := temp * 65536.0;
    IF J < 0 THEN temp := temp + 65536.0 + J ELSE temp :=
      temp + J;
    double_to_real := temp;
  END;
```

```
    {-------------------------------------------------------}
    {    R e a l   t o   D o u b l e    number conversion    }
    {-------------------------------------------------------}
    PROCEDURE Real_to_double(R : real; VAR I, J : integer);
    var It, Jt : real;
    begin
        It := Int(R/65536.0);
        Jt := R - It*65536.0;
        IF It > MaxInt THEN I := trunc(It - 65536.0) ELSE
           I:= trunc(It);
        IF Jt > MaxInt THEN J := trunc(Jt - 65536.0) ELSE
           J:= trunc(Jt);
    END;

    {-------------------------------------------------------}
    {   S e t   T i m e    Turn timer on                    }
    {-------------------------------------------------------}
    PROCEDURE Set_Timer(the_time : integer);
    begin
      tics := double_to_real(HiClock, LoClock);
      tics := tics + the_time*18.206481934;
      real_to_double(tics, timer_hi, timer_lo);
      Status := status or Timer_On;
    END;

PROCEDURE BeBeep;          {makes a nice beep, beep sound}
  VAR N : byte;            {called before message is spoken}
  begin
    nosound;
    FOR N := 1 to 3 do
      begin
        sound(800); delay(50);
        sound(400); delay(50);
      END;
    nosound;
  END;

{*********************************************************}
{   C H A N G E T A B L E                                  }
{*********************************************************}
procedure changetable;    {allows user to change schedule
                             table}
label 10,out;             {label 10 writes schedule and
                           label out gets out of changetable
begin
    clrscr;               {clear the screen}
 10:for i:= 1 to j do     {j is number of table entries}
      begin
        Writeln('Entry #',i:2,'  ',times[i],'  ',mesg[i]);
                          {write table out}
      end;
```

```
Writeln;                {skip a line}
Writeln('Correct?  (Y/N)');        {ask if the
                                    entries are correct}
Writeln;                {skip a line}
Repeat                  {read input until valid}
  Read(kbd,chin);       {do a fast read}
Until chin in ['y','Y','n','N'];   {the valid entries}
If chin in ['n','N'] then      {take action if table is
                                    not correct}
  begin
    writeln('Enter the number of the entry to change
            or');
    writeln('enter FF:FF in an entry''s time field to
            delete the entry or');
    if j < MaxMsg then  {if number of entries is less}
      begin             {than maximum available}
        writeln('enter ',j+1,' to add a new entry to
                the table or');
      end;
    writeln('enter 0 to reaccomplish the entire table
            or');
    writeln('enter 99 to return with no changes.');
    repeat                      {see what the user wants}
      readln(i);                {get input}
      if (i > j+1) and (i <> 99) then
        writeln('Value too high.');
                                {make sure user enters}
      if i < 0 then writeln('Value too low.');
                                {a valid input}
    until i in [0..j+1,99];     {the valid entries}
    if i = 99 then              {user doesn't want any}
      begin                     {changes, go back to
                                 relist}
        i:=1;                   {reinitialize i}
        clrscr;                 {clear the screen}
        goto 10;            {go back to relist schedule}
      end;
    if i = 0 then goto out;  {user wants to rebuild,
                              exit with i=0}
    if i > j then j:=i;      {wants to add an entry,
                              increase table size}
    clrscr;                  {clear the screen}
    gotoxy(15,10);           {set the cursor}
    Writeln('Time ',i,'?  (hh:mm)');  {get new entry or
                                       change the old one}
    gotoxy(15,12);           {set the cursor}
    Readln(times[i]);        {get the time}
    gotoxy(15,14);           {ask the user for the
                              message}
    Writeln('Message ',i,'?    Punctuation required.');
    gotoxy(15,16);           {go to end of line}
```

```
        Readln(mesg[i]);              {read the message}
        k:=length(mesg[i])            {check to make sure
                                       punctuated}
        line:=mesg[i];                {change to string}
        if not (line[k] in ['.','?','!']) then
          begin                       {tell the user to add
                                       punctuation}
            gotoxy(15,20)             {set the cursor}
            writeln('Punctuation (''.'',''?'',or ''!''')
                    required.');
            delay(1500);              {delay to let user read}
            gotoxy(15,20);            {reset the cursor}
            writeln('                                    ');
            gotoxy(15+k,16);          {reset the cursor at end}
            readln(punct);            {get punctuation}
            mesg[i]:=mesg[i]+punct;       {and add it to the
                                           message}
          end;
        said[i]:=false;               {haven't said this yet}
        ClrScr;                       {clear the screen}
        goto 10;                    {go back, print the table,
                                     and see if its correct now}
      out:                          {label to exit procedure}
      end;
end;                                {procedure changetable}

{*********************************************************}
{       G E T D A T A                                     }
{*********************************************************}
Procedure Getdata;                  {initializes table on
                                     entry or if user wants to
                                     reinitialize}

label go,10;

begin
  go:                             {label to start getting data}
  i:=1;                           {initialize variables}
  j:=1;
  clrscr;                         {clear the screen}
  gotoxy(10,3);                   {position the cursor}
  writeln('Enter the time (hh:mm) then the message you wish
           spoken.');
  gotoxy(10,4);                    {position the cursor}
  writeln('Time range is 00:00-23:59.  Message is a maximum
           of 127');
  gotoxy(10,5);                    {position the cursor}
  writeln('characters.  ',MaxMsg,' different messages may
           be entered.');
  writeln;                        {skip a line}
```

```
for i:=1 to MaxMsg do          {read in all message
                                entries}
    begin
      gotoxy(15,10);           {time to get user input}
      Writeln('Time ',i,'?  (hh:mm)      Q to quit.');
      gotoxy(15,12);           {set the cursor}
      Readln(times[i]);        {get time desired for
                                message}
      if (times[i]='q') or (times[i]='Q') then
                               {see if user wants to quit}
         begin
           j:=i-1;             {user wants to quit, set
                                up number of}
           clrscr;             {valid entries in j then
                                call change-}
           goto 10;            {table procedure to list
                                out the entries}
         end;
      gotoxy(15,14);           {get the desired message
                                now}
      Writeln('Message ',i,'?    Punctuation required.');
      gotoxy(15,16);           {read the user's input}
      Readln(mesg[i]);
      k:=length(mesg[i]);      {check to see if
                                punctuated}
      line:=mesg[i];           {change to string}
      if not (line[k] in ['.','?','!']) then
        begin                  {no punctuation, tell the
                                user}
          gotoxy(15,20);       {position the cursor}
          writeln('Punctuation (''.'',''?'',or ''!'')
                   required.');
          delay(1500);         {give user time to read}
          gotoxy(15,20);       {set the cursor}
          writeln('                                    ');
          gotoxy(15+k,16);     {set the cursor at the end}
          readln(punct);       {get the punctuation}
          mesg[i]:=mesg[i]+punct;  {and add it to the
                                    message}
        end;
      said[i]:=false;          {haven't said this yet}
      ClrScr;                  {clear the screen}
      j:=i;                    {entered max entries, so j
                                can equal i}
    end;
10:  changetable;   {procedure changetable lists the
                     entries and let user change}
   if i=0 then goto go;  {if i=0 on return from
                          changetable, user wants to
                          reinitialize}
end;                     {procedure getdata}
```

```
{-------------------------------------------------------}
{      NOW BEGINS THE REAL PROGRAM                      }
{-------------------------------------------------------}

{-------------------------------------------------------}
{                C H E C K                              }
{-------------------------------------------------------}

PROCEDURE check;
   type
     twostr                = String[2];
   var
     timestr               : String[5];
     hrstr                 : twostr;
     minstr                : twostr;
     buffer                : String[127];

Procedure gettime(var hr1,mnt:twostr);
{gets the time in hour and min}
   begin
     tics:=double_to_real(HiClock,LoClock)/18.206481934;
                          {current time ticks}
     str(trunc(tics/3600.0) MOD 24,hr1);    {get hours}
     str(trunc(tics/60) MOD 60, mnt);       {get minutes}
   end;{procedure gettime}

{$I b:speak.inc} {voice interface procedure}

begin
  While Keypressed DO read(Kbd,KeyChr); {clear any waiting
                                         keys}
  IF (status AND timer_on) = timer_on THEN {If our timer is
                                            ticking ..}
    begin
      IF (status AND from_timer) = from_timer THEN {and the
                                        timer finished..}
        begin                           {then clear the
                                         timer request }
          status := status and not (timer_on + from_timer);
          gettime(hrstr,minstr);        {get the current
                                         time};
          if (hrstr[1] = ' ') then hrstr[1]:='0'; {change
                                         blank to zero}
          if (minstr[1] = ' ') then minstr[1]:='0'; {change
                                         blank to zero}
          timestr:=hrstr+':'+minstr;    {concatenate hr
                                        and min}
          for i:= 1 to j do             {check all entries
                                         for a time match}
```

```
                begin
                  if (timestr=times[i]) then {if time matches a
                                              time}
                    begin
                      if not said[i] then     {did we tell the
                                               user already?}
                        begin                 {no, tell the
                                               user}
                          bebeep;             {an attention
                                               getter}
                          speak(mesg[i],65,180); {say the
                                                  message}
                          said[i]:=true;      {set a flag that
                                               we said the msg}
                        end;                  {if not said}
                    end;                      {if (timestr...}
                end;                          {for i:=1 to j...}
              set_timer(timer_time);          {issue another
                                               time call}
            end
          else                                {user must have
                                               entered ALT-F7}
            begin
              MkWin(1,1,80,25,bright+cyan,black,3);
                                              {make a window}
              gotoxy(10,12);                  {set the cursor}
              writeln('Enter R to review/revise schedule or T
                       to terminate.');
              repeat                          {get user input}
                read(kbd,chin);               {do fast read}
              until chin in ['r','R','t','T']; {valid inputs}
              if chin in ['t','T']
                 then terminate:=true   {user wants to cancel}
                 else                   {user wants to
                                         review/revise table}
                   begin
                     i:=1;              {initialize i}
                     changetable;       {user wants the
                                         entries}
                     if i=0 then getdata;  {if i=0 is returned,
                                            then user wants}
                   end;             {to reinitialize the table}
               RmWin;                         {remove the window}
            end;                              {if chin in ['t'...}
        end;                                  {end else}
end;                                          {procedure check}

{-------------------------------------------------------------}
{    THE ABOVE ARE THE USER INCLUDE ROUTINES                  }
{-------------------------------------------------------------}
```

```
{-------------------------------------------------------}
{   P R O C E S S   I N T E R R U P T                   }
{-------------------------------------------------------}
{  PURPOSE:
       The  following  procedures  displace  standard
interrupts.

     Do not put Variables or Constants in this Procedure.
It  will  cause  registers  to  be  clobbered  during  the
Interrupt routine when Turbo attempts to allocate storage
for local variables or parameters.}

PROCEDURE STAY_INT16;        {Keyboard Interrupt 16 Service
                              Routine}
{If anything but "Our_HotKey" is pressed,  the  key  is
passed to the standard  keyboard service routine.  B_U_T,
when Our HotKey is recognized, a hotkey bit is set.}
begin
{$I b:Stayi16.410}
End; {STAY_INT16}

PROCEDURE STAY_INT13;        {BIOS Disk interrupt  Routine}
begin                      {Sets a flag while disk is active}
{$I b:Stayi13.410}
End; {STAY_INT13}

PROCEDURE STAY_INT21;     {DOS interrupt 21 Service Routine}
begin                    {Sets a flag while INT 21 is active}
{$I b:Stayi21.410}
End; {STAY_INT21}

PROCEDURE Stay_INT8;     {Timer Interrupt 8 Service Routine}
                    {Activates Stayres during pgm execution}
begin                        {when safe to do so.}
{$I b:ClkI8.410}

{$I b:Stayi8.420}
End;{Stay_Int8}

PROCEDURE Stay_INT28;  {Idle Interrupt 28 Service Routine}
begin                 {Invokes Stayres from the DOS prompt}
{$I b:Stayi28.410}    {and allows background activity to  }
End;{Stay_Int28}      {continue}

PROCEDURE StaySave;   {Prolog to Resident Turbo Code}
begin
{$I b:StaySave.420}

   GetDTA(SavedDTA[1],SavedDTA[2]);  {Save callers DTA
                                      address}
```

```
   GetPSP(SavedPSP);                    {Save callers PSP
                                         Segment}
   SetPSP(OurPSP);                      {Set our PSP Segment}

   SetDTA(OurDTA[1],OurDTA[2]);         {Set our DTA address}

   NewCtlc[2] := CSeg;
      NewCtlc[1] := Ofs(IRET);
   GetCtlC(SavedCtlc); SetCtlC(NewCtlc);{Get/Save the users
                                         Ctrl-C vector}

   INT24On;                             {Trap Dos Critical
                                         Errors}

        {-------------------------------------------------}
        {        INVOKE USER PROCEDURE HERE               }
        {-------------------------------------------------}

         begin
            KeyChr := #0;           { Clear any residual }
            check;                  {go execute the program}
         end;

        {-------------------------------------------------}
        {   END USER PROCEDURE HERE                       }
        {-------------------------------------------------}

   SetPSP(SavedPSP);                { Restore Callers PSP
                                      Segment}

   SetDTA(SavedDTA[1],SavedDTA[2]);{ Restore the users DTA}

   SetCtlC(SavedCtlC);              { Restore the users Ctrl-C
                                      Vector}

   INT24Off;                        { Remove Our Critical
                                      Error routine}
   If (Terminate = true) then Stay_Xit;{ If exit key,
                                      restore Int Vectors }
{-------------------------------------------------------}
{     BEGINNING OF THE STAYRSTR ROUTINE                 }
{-------------------------------------------------------}
{$I b:Stayrstr.420}      { RETURN TO CALLER }
          {-----------------------------------------------}
          {   END OF THE STAYRSTR ROUTINE                 }
          {-----------------------------------------------}

End ;{StaySave}
```

```
{-------------------------------------------------------}
{                      M  A  I  N                       }
{-------------------------------------------------------}
   { The main program installs the new interrupt routine }
   { and makes it permanently resident as the keyboard   }
   { interrupt.  The old keyboard interrupt Vector is    }
   { stored in Variables , so they can be used in Far    }
   { Calls.                                              }

   { The following dos calls are used:                   }
   { Function 25 - Install interrupt address             }
   {               input al = int number,                }
   {               ds:dx = address to install            }
   { Function 35 - get interrupt address                 }
   {               input al = int number                 }
   {               output es:bx = address in interrupt   }
   { Function 31 - terminate and stay resident           }
   {               input dx = size of resident program   }
   {               obtained from the memory              }
   {               allocation block at [Cs:0 - $10 + 3]  }
   { Function 49 - Free Allocated Memory                 }
   {               input Es = Block Segment to free      }
   {-----------------------------------------------------}

begin                                          {**main**}


  OurDseg:= Dseg;       { Save the Data Segment Address for
                          Interrupts }
  OurSseg:= Sseg;       { Save our Stack Segment for
                          Interrupts        }
  GetPSP(OurPSP);       { Local PSP Segment             }

  GetDTA(OurDTA[1],OurDTA[2]); { Record our DTA address }

  UserProgram:=Ofs(Staysave); {Set target of call
                               instruction}
      Regs.Ax := $3000 ;      {Obtain the DOS Version
                               number}
      Intr(DosI21,Regs);
      DosVersion := Halfregs.Al;  { 0=1+, 2=2.0+, 3=3.0+ }

            {Obtain the DOS Indos status location}
        Regs.Ax := $3400;
        Intr(DosI21,Regs);
        DosStat1.IP  := Regs.BX;
        DosStat1.CS  := Regs.ES;
        DosStat2.CS  := Regs.ES;
        DosSSeg      := Regs.ES;
```

```
    Bytecount := 0;    { Search for CMP [critical flag],00
                         instruction }
    While (Bytecount < $2000)
                       { then Mov SP,stackaddr instruction }
       and (Memw[DosStat2.CS:Bytecount] <> $3E80)
          do Bytecount := Succ(Bytecount);

    If Bytecount = $2000 then begin   { Couldn't find
                                        critical flag addr }
       Writeln('StayRes incompatiblity with Operating
                System');
       Writeln('StayRes will not install
                correctly..Halting');
       Halt; end;

  { Search for the DOS Critical Status Byte address.  }
  { Bytecount contains offset from DosStat1.CS of the }
  {       CMP [critical flag],00                      }
  {       JNZ ....                                    }
  {       Mov SP,indos stack address                  }

    If Mem[DosStat2.CS:Bytecount+7] = $BC {MOV SP,xxxx}
       then begin
       DosStat2.IP := Memw[DosStat2.CS:Bytecount+2];
       DosSptr     := Memw[DosStat2.CS:bytecount+8];
                      {INDOS Stack address}
       END
    else begin
       Writeln('Cannot Find Dos Critical byte...Please
                Reboot.');
       Halt;
       end;

    Inline($FA);                  {Disable interrupts}

    { Setup Our Interrupt Service Routines }

   Setup_Interrupt(BIOSI16, BIOS_Int16, Ofs(Stay_INT16));
     {keyboard}
   Setup_Interrupt(BIOSI8, BIOS_Int8, Ofs(Stay_INT8));
     {timer}
   Setup_Interrupt(BIOSI13, BIOS_Int13, Ofs(Stay_INT13));
     {disk}
   Setup_Interrupt(DOSI21, DOS_Int21, Ofs(Stay_INT21));
     {DOSfunction}
   Setup_Interrupt(DOSI28, DOS_Int28, Ofs(Stay_INT28));
     {DOS idle}

    Inline($FB);                  {Re-enable interrupts}
{*********************************************************}
```

```
{-------------------------------------------------------}
{            INITIALIZE YOUR PROGRAM HERE               }
{-------------------------------------------------------}
{*******************************************************}
{ Initialize Program Here since we will not get control
again.}

Terminate := false;   {Clear the program exit flags }
MkWin(1,1,80,25,bright+cyan,black,3);  {make a window}
clrscr;
getdata;              {set up initial times and msgs}
RmWin;
writeln;
writeln('***********************************************');
writeln('***     Remind System is now resident.      ***');
writeln('*** Enter ALT-F7 to review/revise schedule  ***');
writeln('***          or terminate program.          ***');
writeln('***********************************************');
set_timer(timer_time);          {start the timer}
     {-----------------------------------------}
     {    END OF INITALIZE PROGRAM CODE        }
     {-----------------------------------------}
{  Now terminate and stay resident.  The following Call
utilizes the DOS Terminate & Stay Resident function.  We
get the amount of memory by fetching the memory allocation
paragraphs from the Memory Control Block.  This was set by
Turbo initialization during Int 21/function 4A (shrink
block), calculated from the mInimum and mAximum options
menu. The MCB sits one paragraph above the PSP.}
                  { Pass return code of zero    }
     Regs.Ax := $3100 ;    { Terminate and Stay Resident }
     Regs.Dx := MemW [Cseg-1:0003]+1 ;   { Prog_Size from
                                           Allocation Blk}
     Intr (DosI21,Regs);


       { END OF RESIDENCY CODE }
end.
```

Files that are included in the above program are listed below. The procedure **SPEAK.INC** can be found in the *IBM Text-to-Speech Interface for the IBM Voice Communications Adapter*, Talbot, Summer 1987.

```
{************************************************************}
{           S T A Y W N D O . 3 4 1                          }
{           "...but I dont do floors !"                      }
{************************************************************}
{   Kloned and Kludged by Lane Ferris                        }
{       -- The Hunters Helper --                             }
{ Original Copyright 1984 by Michael A. Covington            }
{ Modifications by Lynn Canning 9/25/85                      }
{   1) Foreground and Background colors added.               }
{      Monochrome monitors are automatically set             }
{      to white on black.                                    }
{   2) Multiple borders added.                               }
{   3) TimeDelay procedure added.                            }
{ Requirements: IBM PC or close compatible.                  }
{------------------------------------------------------------}
{ To make a window on the screen, call the procedure}
    {MkWin(x1,y1,x2,y2,FG,BG,BD);
{   The x and y coordinates define the window placement and
are the same as the Turbo Pascal Window coordinates.  The
border parameters (BD) are 0 = No border   1 = Single line
border      2 = Double line border  3 = Double Top/Bottom
Single sides  }

The foreground (FG) and background (BG) parameters are the
same values as the corresponding Turbo Pascal values.}

{ The maximum number of windows open at one time is set at
five see MaxWin=5).  This may be set to greater values if
necessary.}

{ After the window is made, you must write the text desired
from the calling program.  Note that the usable text area
is actually 1 position smaller than the window coordinates
to allow for the border.  Hence, a window defined as
1,1,80,25 would actually be 2,2,79,24 after the border is
created.  When writing to the window in your calling
program, the textcolor and backgroundcolor may be changed
as desired by using the standard Turbo Pascal
commands.                }

{ To return to the previous screen or window, call the
procedure RmWin; }

{ The TimeDelay procedure is invoked from your calling
program.  It is similar to the Turbo Pascal DELAY except
DELAY is based on clock speed whereas TimeDelay is based on
the actual clock.  This means that the delay will be the
same duration on all systems no matter what the clock
speed.  The procedure could be used for an error condition
as follows:       }
```

```
{   MkWin          - make an error message window          }
{   Writeln        - write error message to window         }
{   TimeDelay(5)   - leave window on screen 5 seconds      }
{   RmWin          - remove error window                   }
{   continue processing                                    }
{----------------------------------------------}

Const

      InitDone :boolean = false ; { Initialization switch}

      On      = True ;
      Off     = False ;
      VideoEnable = $08;          { Video Signal Enable Bit }
      Bright = 8;                 { Bright Text bit}
      Mono   = 7;                 {MonoChrome Mode}

Type
     Imagetype  = array [1..4000] of char;  { Screen Image
                                              in the heap}
     WinDimtype = record
                    x1,y1,x2,y2: integer
                  end;

     Screens    = record          { Save Screen Information}
                   Image: Imagetype; { Saved screen Image }
                   Dim:   WinDimtype; { Saved Window
                                        Dimensions }
                   x,y:   integer;{ Saved cursor position }
                  end;


 Var

  Win:                             { Global variable package }
    record
      Dim:    WinDimtype;      { Current Window Dimensions }
      Depth:  integer;
          { MaxWin should be included in your program }
          { and it should be the number of windows
            saved at one time }
      { It should be in the const section of your program }
      Stack:  array[1..MaxWin] of ^Screens;
    end;

  Crtmode     :byte      absolute $0040:$0049;
                                {Crt Mode,Mono,Color,B&W..}
  Crtwidth    :byte      absolute $0040:$004A;
                                {Crt Mode Width, 40:80 .. }
  Monobuffer  :Imagetype absolute $B000:$0000;
                                {Monochrome Adapter Memory}
```

```
  Colorbuffer :Imagetype absolute $B800:$0000;
                                  {Color Adapter Memory     }
  CrtAdapter  :integer   absolute $0040:$0063;
                                  { Current Display Adapter }
  VideoMode   :byte      absolute $0040:$0065;
                                  { Video Port Mode byte    }
  TurboCrtMode: byte     absolute  Dseg:6;
                                  {Turbo's Crt Mode byte    }
   Video_Buffer:integer;          { Record the current Video}
  Delta,
  x,y         :integer;

{-------------------------------------------------------}
{         Delay for  X seconds                          }
{-------------------------------------------------------}

procedure TimeDelay (hold : integer);
type
  RegRec =                    { The data to pass to DOS }
    record
      AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
    end;
var
  regs:regrec;
  ah, al, ch, cl, dh:byte;
  sec                :string[2];
  result, secn, error, secn2, diff :integer;

begin
  ah := $2c;                      {Get Time-Of-Day from DOS}
  with regs do                    {Will give back Ch:hours  }
                                  {Cl:minutes,Dh:seconds    }
    ax := ah shl 8 + al;          {Dl:hundreds              }
  intr($21,regs);

  with regs do
    str(dx shr 8:2, sec);              {Get seconds      }
                                       {with leading null}
  if (sec[1] = ' ') then
    sec[1]:= '0';
  val(sec, secn, error);          {Conver seconds to integer}
  repeat                  { stay in this loop until the time}
     ah := $2c;                        { has expired }
     with regs do
        ax := ah shl 8 + al;
     intr($21,regs);                   {Get current time-of-day}

     with regs do                      {Normalize to Char}
        str(dx shr 8:2, sec);
     if (sec[1] = ' ') then
        sec[1]:= '0';
```

```
      val(sec, secn2, error);  {Convert seconds to integer}
      diff := secn2 - secn;     {Number of elapsed seconds}
      if diff < 0 then      { we just went over the minute }
         diff := diff + 60;       { so add 60 seconds }
  until diff > hold;        { has our time expired yet }
end; { procedure TimeDelay }

{-------------------------------------------------------}
{   Get Absolute postion of Cursor into parameters x,y }
{-------------------------------------------------------}
Procedure Get_Abs_Cursor (var x,y :integer);
  Var
      Active_Page  : byte absolute $0040:$0062;
                          { Current Video Page Index}
Crt_Pages  : array[0..7] of integer absolute $0040:$0050 ;

   Begin
      X := Crt_Pages[active_page];   {Get Cursor Position }
      Y := Hi(X)+1;                 { Y get Row}
      X := Lo(X)+1;                 { X gets Col position}
   End;

{-------------------------------------------------------}
{   Turn the Video On/Off to avoid Read/Write snow      }
{-------------------------------------------------------}
Procedure Video (Switch:boolean);
   Begin
      If (Switch = Off) then
      Port[CrtAdapter+4] := (VideoMode - VideoEnable)
      else Port[CrtAdapter+4] := (VideoMode or
                                  VideoEnable);
   End;
{-------------------------------------------------------}
{ InitWin Saves the Current (whole) Screen              }
{-------------------------------------------------------}
Procedure InitWin;
  { Records Initial Window Dimensions }
   Begin
     with Win.Dim do
       begin x1:=1; y1:=1; x2:=crtwidth; y2:=25 end;
     Win.Depth:=0;
     InitDone := True ;       { Show initialization Done }
end;
{-------------------------------------------------------}
{ BoxWin Draws a Box around the current Window          }
{-------------------------------------------------------}
procedure BoxWin(x1,y1,x2,y2, BD, FG, BG :integer);

{Draws a box, fills it with blanks, and makes it the
current Window.  Dimensions given are for the box; actual
Window is one unit smaller in each direction.  }
```

```
var
    I,
    TB,SID,TLC,TRC,BLC,BRC    :integer;

begin
  if Crtmode = Mono then begin
    FG := 7;
    BG := 0;
    end;
  Window(x1,y1,x2,y2);                    {Make the Window}
  TextColor(FG) ;                         {Set the colors}
  TextBackground(BG);
  Case BD of                            {Make Border characters}
    0:;                                 {No border option}
    1:begin                             {Single line border option}
      TB  := 196;                             {Top Border}
      SID := 179;                             {Side Border}
      TLC := 218;                             {Top Left Corner}
      TRC := 191;                             {Top Right Corner}
      BLC := 192;                             {Bottom Left Corner}
      BRC := 217;                             {Bottom Right Corner}
      end;
    2:begin                             {Double line border option}
      TB  := 205;
      SID := 186;
      TLC := 201; TRC := 187;
      BLC := 200; BRC := 188;
      end;
    3:begin              {Double Top/Bottom with single sides}
      TB  := 205;            {"deary and dont spare the lace"}
      SID := 179;
      TLC := 213; TRC := 184;
      BLC := 212; BRC := 190;
      end;
    End;{Case}

  IF BD > 0 then begin              { User want a border? }
  { Top }
     gotoxy(1,1);                   { Window Origin       }
     Write( chr(TLC) );             { Top Left Corner     }
     For I:=2 to x2-x1   do         { Top Bar             }
        Write( chr(TB));
     Write( chr(TRC) );             { Top Right Corner    }

  { Sides  }
     for I:=2 to y2-y1 do begin
         gotoxy(1,I);                { Left Side Bar       }
         write( chr(SID) );
         gotoxy(x2-x1+1,I) ;         { Right Side Bar      }
         write( chr(SID) );
       end;
```

```
  { Bottom }
     gotoxy(1,y2-y1+1);              { Bottom Left Corner }
     write( chr(BLC) );
     for I:=2 to x2-x1   do          { Bottom Bar         }
        write( chr(TB) );

  { Make it the current Window }
     Window(x1+1,y1+1,x2-1,y2-1);
     write( chr(BRC) );             { Bottom Right Corner }
   end; {If BD > 0};

   gotoxy(1,1) ;
   TextColor( FG) ;               { Take Low nibble 0..15  }
   TextBackground (BG);           { Take High nibble  0..9 }
   ClrScr;
 end;
{-------------------------------------------------------------}
{ MkWin   Make a Window                                       }
{-------------------------------------------------------------}
procedure MkWin(x1,y1,x2,y2, FG, BG, BD :integer);
  { Create a removable Window }

begin

  If (InitDone = false) then { Initialize if not done yet }
      InitWin;

  TurboCrtMode := CrtMode;   {Set Textmode w/o ClrScr}
  If CrtMode = 7 then Video_Buffer := $B000 {Set Ptr to
                                             Monobuffer}
  else  Video_Buffer := $B800;          {or Color Buffer }

  with Win do Depth:=Depth+1;   { Increment Stack pointer }
  if Win.Depth>maxWin then
    begin
      writeln(^G,' Windows nested too deep ');
      halt
    end;
                {--------------------------------------}
                {       Save contents of screen        }
                {--------------------------------------}
  With Win do
    Begin
    New(Stack[Depth]);  { Allocate Current Screen to Heap }
    Video( Off);
    If CrtMode = 7 then
    Stack[Depth]^.Image := monobuffer { set pointer to it }
    else
    Stack[Depth]^.Image := colorbuffer ;
    Video( On);
   End ;
```

```
  With Win do
     Begin                              { Save Screen Dimentions}
     Stack[Depth]^.Dim := Dim;
     Stack[Win.Depth]^.x  := wherex;{ Save Cursor Position}
     Stack[Win.Depth]^.y  := wherey;
     End ;
                              { Validate the Window Placement}
  If (X2 > 80) then           { If off right of screen       }
          begin
          Delta := (X2 - 80);  { Overflow off right margin}
          If X1 > Delta then
             X1 := X1 - Delta ; { Move Left window edge }
          X2 := X2 - Delta ;    {Move Right edge on 80   }
          end;
  If (Y2 > 25) then           { If off bottom  screen        }
          begin
          Delta := Y2 - 25; { Overflow off right margin }
          If Y1 > Delta then
             Y1 := Y1 - Delta ;  { Move Top edge up}
           Y2 := Y2 - Delta ;       { Move Bottom  24 }
          end;
                                    { Create the New Window  }
  BoxWin(x1,y1,x2,y2,BD,FG,BG);
  If BD >0 then begin          {Shrink window within borders}
     Win.Dim.x1 := x1+1;
     Win.Dim.y1 := y1+1;      { Allow for margins }
     Win.Dim.x2 := x2-1;
     Win.Dim.y2 := y2-1;
     end;
end;
{-------------------------------------------------------------}
{                  Remove Window                              }
{-------------------------------------------------------------}
    { Remove the most recently created removable Window }
    { Restore screen contents, Window Dimensions, and   }
    { position of cursor.  }
Procedure RmWin;
  Var
    Tempbyte : byte;

   Begin
   Video(Off);
   With Win do
      Begin                     { Restore next Screen        }
      If crtmode = 7 then
      monobuffer := Stack[Depth]^.Image
      else
      colorbuffer := Stack[Depth]^.Image;
      Dispose(Stack[Depth]);  { Remove Screen from Heap    }
      end;
```

```
    Video(On);

    With Win do                    { Re-instate the Sub-Window }
     Begin                         { Position the old cursor   }
       Dim := Stack[Depth]^.Dim;
       Window(Dim.x1,Dim.y1,Dim.x2,Dim.y2);
       gotoxy(Stack[Depth]^.x,Stack[Depth]^.y);
     end;

       Get_Abs_Cursor(x,y) ;      {New Cursor Position }
       Tempbyte :=              { Get old Cursor attributes}
            Mem[ Video_Buffer:((x-1 + (y-1) * 80 ) * 2)+1 ];

     TextColor( Tempbyte And $0F );{ Take Low nibble  0..15}
     TextBackground ( Tempbyte Div 16); { Take High nibble
                                          0..9}
       Depth := Depth - 1
     end ;
end;
{-------------------------------------------------------}
{*******************************************************}
{   S T A Y X I T   .   4 2 0                           }
{*******************************************************}
{-------------------------------------------------------}
{Stay_Xit Check Terminate Keys                          }
{                                                       }
{  Clean up the Program ,Free the Environment block, the
program segment memory and return to Dos. Programs using
this routine ,must be the last program in memory, else ,a
hole  will  be  left  causing  Dos  to  take  off  for
Peoria.                                                 }
{-------------------------------------------------------}
Procedure Stay_Xit;
{
       This code reinstates those interrupts that will not
be restored by DOS  Interrupts 22,23,24 (hex) are restored
from the Current PSP during termination.}
  VAR
   PSPvector22: vector absolute Cseg:$0A;
   PSPvector23: vector absolute Cseg:$0E;
   PSPvector24: vector absolute Cseg:$12;

   DOSvector22: vector absolute 0:$88;
   DOSvector23: vector absolute 0:$8C;
   DOSvector24: vector absolute 0:$90;

    Begin { Block }
      writeln;
      Writeln ('Remind program Terminated') ;
      WRITELN;
      WRITELN ('Enter <CR> to continue');
```

```
        Inline($FA);                          {Disable interrupts}

            { Restore Disk Interrupt Service Routine  }

        Regs.Ax := $2500 + BIOSI13;
        Regs.Ds := BIOS_INT13.CS;
        Regs.Dx := BIOS_INT13.IP;
        Intr ($21,Regs);

              { Restore Keyboard Interrupt Service Routine  }

        Regs.Ax := $2500 + BIOSI16;
        Regs.Ds := BIOS_INT16.CS;
        Regs.Dx := BIOS_INT16.IP;
        Intr ($21,Regs);

              { Restore Timer Interrupt Service Routine  }

        Regs.Ax := $2500 + BIOSI8;
        Regs.Ds := BIOS_INT8.CS;
        Regs.Dx := BIOS_INT8.IP;
        Intr ($21,Regs);

              { Restore DOS 21 Interrupt Service Routine  }

        Regs.Ax := $2500 + DOSI21;
        Regs.Ds := DOS_INT21.CS;
        Regs.Dx := DOS_INT21.IP;
        Intr ($21,Regs);

              { Restore DOS 28 Interrupt Service Routine  }

        Regs.Ax := $2500 + DOSI28;
        Regs.Ds := DOS_INT28.CS;
        Regs.Dx := DOS_INT28.IP;
        Intr ($21,Regs);

    { Move Interrupt Vectors 22,23,24 to our PSP from where
DOS will restore }

        PSPvector22 := DOSvector22;      { Terminate vector }
        PSPvector23 := DOSvector23;      { Cntrl-C vector   }
        PSPvector24 := DOSvector24;      { Critical vector  }

        Inline($FB);                     {Re-enable interrupts}

        Regs.Ax := $49 shl 8 + 0 ; { Free Allocated Block
                                     function}
        Regs.Es := MemW[Cseg:$2C]; { Free environment
                                     block       }
        MsDos( Regs ) ;
```

```
      Regs.Ax := $49 shl 8 + 0;  { Free Allocated Block
                                   function}
      Regs.Es := Cseg ;           { Free Program}
      MsDos( Regs ) ;
   End  { StayXit };

{*********************************************************}
{          S T A Y S U B S  .  4 2 0                      }
{*********************************************************}
      {-----------------------------------------------------}
      {     S E T U P   I N T E R R U P T                   }
      {-----------------------------------------------------}
         Msg # *48   Dated 07-07-86 16:54:36
         From: NEIL RUBENKING
         To: LANE FERRIS
         Re: STAY, WON'T YOU?

         Lane,
              Here's what I did:
}
    PROCEDURE   Setup_Interrupt(IntNo   :byte;   VAR   IntVec
:vector; offset :integer);
  BEGIN
    Regs.Ax := $3500 + IntNo;
    Intr(DosI21,Regs);      {get the address of interrupt }
    IntVec.IP := Regs.BX;  { Location of Interrupt Ip }
    IntVec.CS := Regs.Es;  { Location of Interrupt Cs }

    Regs.Ax := $2500 + IntNo; { set the interrupt to point
                                to our procedure}
    Regs.Ds := Cseg;
    Regs.Dx := Offset;
    Intr (DosI21,Regs);
  END;
(********C O M M E N T *************
{in the main part of the program}
Setup_Interrupt(BIOSI16, BIOS_Int16, Ofs(Stay_INT16));
 {keyboard}
Setup_Interrupt(BIOSI10, BIOS_Int10, Ofs(Stay_INT10));
 {video}
Setup_Interrupt(BIOSI8, BIOS_Int8, Ofs(Stay_INT8));
 {timer}
Setup_Interrupt(BIOSI13, BIOS_Int13, Ofs(Stay_INT13));
 {disk}
Setup_Interrupt(DOSI21, DOS_Int21, Ofs(Stay_INT21));
 {DOSfunction}
Setup_Interrupt(DOSI28, DOS_Int28, Ofs(Stay_INT28));
 {DOS idle}
******C O M M E N T ******************)
```

```
{-------------------------------------------------------}
{                  S E T    D T A                       }
{-------------------------------------------------------}
   Procedure SetDTA(var segment, offset : integer );
   BEGIN
     regs.ax := $1A00;{ Function used to get current DTA
                        address}
     regs.Ds := segment;     { Segment of DTA returned by
                               DOS }
     regs.Dx := offset;      { Offset of DTA returned }
     MSDos( regs );       { Execute MSDos function request }
   END;
      {-------------------------------------------------}
      {           G E T    D T A                       }
      {-------------------------------------------------}
   Procedure GetDTA(var segment, offset : integer );
   BEGIN
     regs.ax := $2F00;       { Function used to get current
                               DTA address }
     MSDos( regs );          { Execute MSDos function
                               request }
     segment := regs.ES;     { Segment of DTA returned by
                               DOS }
     offset  := regs.Bx;     { Offset of DTA returned }
   END;
      {-------------------------------------------------}
      {           S E T    P S P                       }
      {-------------------------------------------------}
   Procedure SetPSP(var segment : integer );
   BEGIN
       { A bug in DOS 2.0, 2.1, causes DOS to clobber its
standard stack when the PSP get/set functions are issued at
the DOS prompt. The following checks are made, forcing DOS
to use the "critical" stack when the TSR enters at the
INDOS level.}

   {If Version less then 3.0 and INDOS set }
   If DosVersion < 3 then {then set the Dos Critical Flag}
      If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
         Mem[DosStat2.CS:DosStat2.IP] := $FF;
  regs.ax := $5000; { Function to set current PSP address }
  regs.bx := segment; { Segment of PSP to be used by DOS }
  MSDos( regs );          { Execute MSDos function request }
 {If Version less then 3.0 and INDOS set }
 If DosVersion < 3 then {then clear the Dos Critical Flag }
     If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
        Mem[DosStat2.CS:DosStat2.IP] := $00;
END;
```

```
      {-------------------------------------------------}
      {          G E  T    P  S  P                      }
      {-------------------------------------------------}
   Procedure GetPSP(var segment : integer );
   BEGIN
        { A bug in DOS 2.0, 2.1, causes DOS to clobber its
standard stack when the PSP get/set functions are issued at
the DOS prompt. The following checks are made, forcing DOS
to use the "critical" stack when the TSR enters at the
INDOS level. }
     {If Version less then 3.0 and INDOS set }
   If DosVersion < 3 then { then set the Dos Critical Flag}
      If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
          Mem[DosStat2.CS:DosStat2.IP] := $FF;

    regs.ax := $5100;{Function to get current PSP address }
     MSDos( regs );    { Execute MSDos function request }
     segment := regs.Bx; { Segment of PSP returned by DOS }

     {IF DOS Version less then 3.0 and INDOS set }
If DosVersion < 3 then  {then clear the Dos Critical Flag }
      If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
          Mem[DosStat2.CS:DosStat2.IP] := $00;
   END;
    {-------------------------------------------------}
    {   G e t   C o n t r o l  C (break)  V e c t o r    }
    {-------------------------------------------------}
Type
    Arrayparam = array [1..2] of integer;
Const
     SavedCtlC: arrayparam = (0,0);
     NewCtlC  : arrayparam = (0,0);
 Procedure GetCtlC(Var SavedCtlC:arrayparam);
    Begin              {Record the Current Ctrl-C Vector}
       With Regs Do
       Begin
       AX:=$3523;
       MsDos(Regs);
       SavedCtlC[1]:=BX;
       SavedCtlC[2]:=ES;
       End;
    End;


    {-------------------------------------------------}
    {   S e t   C o n t r o l  C   V e c t o r        }
    {-------------------------------------------------}
    Procedure IRET;          {Dummy Ctrl-C routine}
       Begin
       inline($5D/$5D/$CF);  {Pop Bp/Pop Bp/Iret}
       end;
```

```
 Procedure SetCtlC(Var CtlCptr:arrayparam);
    Begin                          {Set the New Ctrl-C Vector}
       With Regs Do
       Begin
        AX:=$2523;
        DS:=CtlCptr[2];
        DX:=CtlCptr[1];
        MsDos(Regs);
       End;
    End;
{-------------------------------------------------------}
{   K e y i n   :   R e a d  K e a b o a r d           }
{-------------------------------------------------------}
Function Keyin: char;       { Get a key from the Keyboard  }
   Var Ch : char;          { If extended key, fold above 127}
   Begin                   {------------------------------}
      Repeat until Keypressed;
      Read(Kbd,Ch);
      if (Ch = Esc) and KeyPressed then
         Begin
         Read(Kbd,Ch);
         Ch := Char(Ord(Ch) + 127);
         End;
      Keyin := Ch;
   End;  {Keyin}
{-------------------------------------------------------}
{  B e e p   :  S o u n d  t h e  H o r n               }
{-------------------------------------------------------}
Procedure Beep(N :integer); {--------------------------}
   Begin      {  This routine sounds a tone of frequency }
      Sound(n); {  N for approximately 100 ms            }
      Delay(100);          {--------------------------}
      Sound(n div 2);
      Delay(100);
      Nosound;
      End {Beep} ;


      {-------------------------------------------------}
      {      I N T E R R U P T    2 4                   }
      {-------------------------------------------------}
{ Version 2.0, 1/28/86
  -  Bela Lubkin
     CompuServe 76703,3015

     Apologetically mangled by Lane Ferris
```

For MS-DOS version 2.0 or greater, Turbo Pascal 1.0 or greater.

Thanks to Marshall Brain for the original idea for these routines. Thanks to John Cooper for pointing out a small

flaw in the code. These routines provide a method for Turbo Pascal programs to trap MS-DOS interrupt 24 (hex). INT 24h is called by DOS when a 'critical error' occurs, and it normally prints the familiar "Abort, Retry, Ignore?" message.

With the INT 24h handler installed, errors of this type will be passed on to Turbo Pascal as an error. If I/O checking is on, this will cause a program crash. If I/O checking is off, IOResult will return an error code. The global variable INT24Err will be true if an INT 24h error has occurred. The variable INT24ErrorCode will contain the INT 24h error code as given by DOS. These errors can be found in the DOS Technical Reference Manual.

It is intended that INT24Result be used in place of IOResult. Calling INT24Result clears IOResult. The simple way to use INT24Result is just to check that it returns zero, and if not, handle all errors the same. The more complicated way is to interpret the code. The integer returned by INT24Result can be looked at as two bytes. By assigning INT24Result to a variable, you can then examine the two bytes: (Hi(<variable>)-1) will give the DOS critical error code, or (<variable> And $FF00) will return an integer from the table listed in the INT24Result procedure (two ways of looking at the critical error); Lo(<variable>) will give Turbo's IOResult. A critical error will always be reflected in INT24Result, but the IOResult part of INT24Result will not necessarily be nonzero; in particular, unsuccessful writes to character devices will not register as a Turbo I/O error.

INT24Result should be called after any operation which might cause a critical error, if Turbo's I/O checking is disabled. If it is enabled, the program will be aborted except in the above noted case of writes to character devices.

Also note that different versions of DOS and the BIOS seem to react to printer errors at vastly different rates. Be prepared to wait a while for anything to happen (in an error situation) on some machines. These routines are known to work correctly with:
Turbo Pascal 1.00B PC-DOS; Turbo Pascal 2.00B PC-DOS; Turbo Pascal 2.00B MS-DOS;
Turbo Pascal 3.01A PC-DOS. Other MS-DOS and PC-DOS versions should work.

Note that Turbo 2.0's normal IOResult codes for MS-DOS DO

```
NOT correspond to the I/O error numbers given in Appendix
I of the Turbo 2.0 manual, or to the error codes given in
the I/O error nn, PC=aaaa/Program aborted message. Turbo
3.0 IOResult codes do match the manual. Here is a table
of the correspondence (all numbers in hexadecimal): Turbo
2.0 IOResult    Turbo error, Turbo 3.0 IOResult----------
---------------------------------
     00                    00  none
     01                    90  record length mismatch
     02                    01  file does not exist
     03                    F1  directory is full
     04                    FF  file disappeared
     05                    02  file not open for input
     06                    03  file not open for output
     07                    99  unexpected end of file
     08                    F0  disk write error
     09                    10  error in numeric format
     0A                    99  unexpected end of file
     0B                    F2  file size overflow
     0C                    99  unexpected end of file
     0D                    F0  disk write error
     0E                    91  seek beyond end of file
     0F                    04  file not open
     10                    20  operation not allowed on a
                               logical device
     11                    21  not allowed in direct mode
     12                    22  assign to standard files is not
                               allowed
     --                    F3  Too many open files

   -  Bela Lubkin
      CompuServe 76703,3015     1/28/86}

Const
  INT24Err: Boolean=False;
  INT24ErrCode: Byte=0;
  OldINT24: Array [1..2] Of Integer=(0,0);

Var
  RegisterSet: Record Case Integer Of
                 1: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:
                    Integer);
                 2: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
               End;

Procedure INT24;      { Interrupt 24 Service Routine }
  Begin
          Inline(       $2E/$C6/$06/       Int24Err       /
$01/$50/$89/$F8/$2E/$A2/ Int24ErrCode
           /$58/$B0/$00/$89/$EC/$5D/$CF);
```

```
{   Turbo:  PUSH BP              Save caller's stack frame
            MOV  BP,SP          Set up this procedure's stack
                                 frame
            PUSH BP              ?
    Inline:
            MOV  BYTE CS:[INT24Err],1   Set INT24Err to
                                        True
            PUSH AX
            MOV  AX,DI                Get INT 25h error code
            MOV  CS:[INT24ErrCode],AL Save it in
                                       INT24ErrCode
            POP  AX
            MOV  AL,0           Tell DOS to ignore the error
            MOV  SP,BP                 Unwind stack frame
            POP  BP
            IRET                Let DOS handle it from here
}
  End;

       {-------------------------------------------------}
       { I N T  2 4   O N                                }
       {-------------------------------------------------}
  { Grab the Critical error ptr from the previous user}
Procedure INT24On;  { Enable INT 24h trapping }
  Begin
    INT24Err:=False;
    With RegisterSet Do
     Begin
      AX:=$3524;
      MsDos(RegisterSet);
      If (OldINT24[1] Or OldINT24[2])=0 Then
       Begin
        OldINT24[1]:=ES;
        OldINT24[2]:=BX;
       End;
      DS:=CSeg;
      DX:=Ofs(INT24);
      AX:=$2524;
      MsDos(RegisterSet);
     End;
  End;
       {-------------------------------------------------}
       {        I N T  2 4   O F F                       }
       {-------------------------------------------------}
{Give Critical Error Service pointer back to previous user}
Procedure INT24Off;
```

```
  Begin
    INT24Err:=False;
    If OldINT24[1]<>0 Then
      With RegisterSet Do
       Begin
        DS:=OldINT24[1];
        DX:=OldINT24[2];
        AX:=$2524;
        MsDos(RegisterSet);
       End;
    OldINT24[1]:=0;
    OldINT24[2]:=0;
  End;

Function INT24Result: Integer;
  Var
    I:Integer;

  Begin
    I:=IOResult;
    If INT24Err Then
     Begin
      I:=I+256*Succ(INT24ErrCode);
      INT24On;
     End;
    INT24Result:=I;
  End;

{ INT24Result returns all the regular Turbo IOResult codes
if no critical error has occurred.  If a critical error,
then the following values are  added to the error code from
Turbo:
   256:  Attempt to write on write protected disk
   512:  Unknown unit                      [internal dos error]
   768:  Drive not ready                   [drive door open or
                                            bad drive]
  1024:  Unknown command                   [internal dos error]
  1280:  Data error (CRC)                  [bad sector or drive]
  1536:  Bad request structure length      [internal dos error]
  1792:  Seek error                        [bad disk or drive]
  2048:  Unknown media type                [bad disk or drive]
  2304:  Sector not found                  [bad disk or drive]
  2560:  Printer out of paper              [anything that the
                                           printer might signal]
  2816:  Write fault                       [character device not
                                            ready]
  3072:  Read fault                        [character device not
                                            ready]
  3328:  General failure                   [several meanings]
  If you need the IOResult part, use
   I:=INT24Result and 255; [masks out the INT 24h code]
```

```
  For the INT 24h code, use
   I:=INT24Result Shr 8;   {same as Div 256, except faster}
   INT24Result clears both error codes, so you must assign
it to a variable if  you want to extract both codes:
   J:=INT24Result;
   WriteLn('Turbo IOResult  = ',J And 255);
   WriteLn('DOS INT 24h code = ',J Shr 8);

  Note that in most cases, errors on character devices (LST
and AUX) will not  return an IOResult, only an INT 24h
error code. }
{ Main program.  Delete next line to enable }
      {-------------------------------------------------}
      {       G E T    E R R O R    C O D E             }
      {-------------------------------------------------}
   Procedure GetErrorCode;
    Begin
    Error := IOresult;              {Read the I/O result}
    If INT24Err Then
     Begin
      Error:=Error+256*Succ(INT24ErrCode);
      INT24On;
     End;
    Good := (Error = 0);            {Set Boolean Result }
  End;


{*******************************
*   S T A Y I 1 6 . 4 1 0     *
*******************************}

Inline(

{;*****************************************************;}
{;P R O C E S S    I N T E R R U P T   1 6              ;}
{;*****************************************************;}
                                {; Function:}
{;Provide a Keyboard trap to allow concurrent processes to
run in the background while a Turbo Read is active.
{;      Copyright (C) 1985,1986}
{;             Lane Ferris}
{;       - The Hunter's Helper -}
{;       Distributed to the Public Domain for use without
profit.  Original Version5.15.85}
{;           ; On entry the Stack will already contain: ;}
{;           ;  1) Sp for Dos                           ;}
{;           ;  2) Bp for Dos                           ;}
{;           ;  3) Ip for Dos                           ;}
{;           ;  4) Cs for Dos                           ;}
{;           ;  5) Flags for Dos                        ;}
```

```
  $5D
                          {Pop     Bp}
/$5D
                          {Pop     Bp                ;
                           Restore Original Bp}
/$80/$FC/$00
                          {Cmp     Ah,00             ;
                           If Char request,}
/$74/$2A
                         { Je      Func00            ;
                           loop for character}
/$80/$FC/$01
                          {Cmp     Ah,01             ;
                           If character availability test}
/$74/$05
                           {     Je      Func01       ;
                           go check for char}
                           {GoBios16:}
/$2E
                        {  CS:                       ;
                           Go to Bios Interrupt 16}
/$FF/$2E/>BIOS_INT16
                        {  Jmp Far [>BIOS_Int16]}
                           {Func01:}
/$E8/$3F/$00
                         { Call    KeyStat           ;
                           Look at Key buffer}
/$9C                     { PushF}
/$74/$16
                        {  Jz      Fret01            ;
                           Return if no key}
/$2E
                        {  CS:                       ;
                           Test for HOT KEY}
/$3A/$26/>OUR_HOTKEY
                        {  Cmp     Ah,[<Our_HotKey]}
/$75/$0F                   Jne     Fret01}
/$B4/$00
                    {      Mov     Ah,0              ;
                           Remove the HotKey}
/$2E
                    {      CS:                       ;
                    flags are removed by BIOS return}
/$FF/$1E/>BIOS_INT16
                    {      Call Dword [>BIOS_INT16]}
/$2E
                        {  CS:                       ;
                           Say we saw the HOT Key}
/$80/$0E/>STATUS/<HOTKEY_ON
                    {      Or by [<Status],<HotKey_ON}
```

```
/$EB/$E4
                    {     Jmp     Func01          ;}
                          {Fret01:}
/$9D                      {        POPF}
/$CA/$02/$00
                     {    RETF    2               ;
                          Return to user}
                          {Func00:}
/$E8/$1F/$00
                     {    Call    KeyStat         ;
                          Wait until character available}
/$74/$FB             {    Jz      Func00}
/$B4/$00             {    Mov     Ah,0            ;
                          Get the next User Key}
/$9C
                     {    PUSHF                   ;}
/$2E                      {        CS:}
/$FF/$1E/>BIOS_INT16
                     {    Call Dword [>BIOS_INT16]}
/$9C
                     {    PushF                   ;
                          Save Return Flags}
/$2E                 {    CS:}
/$3A/$26/>OUR_HOTKEY
        {         Cmp     Ah,[<Our_HotKey]; Our HotKey ?}
/$74/$04
                     {    Je      GotHotKey       ;
                          yes..enter Staysave code}
/$9D
                    {     POPF                    ;
                          else Restore INT 16 flags}
/$CA/$02/$00
                     {    RetF    2               ;
              Return W/Key discard original INT 16 flags}
             {; ".. give it to Mikey..he'll eat anything"}
                      {GotHotKey:}
/$9D                  { POPF ; Discard INT16 return flags}
/$2E         { CS:        ; Say we saw the HOT Key}
/$80/$0E/>STATUS/<HOTKEY_ON { Or by [<Status],<HotKey_ON}
/$EB/$DE                  {Jmp  Func00; Get another Key}
                          {;}
{;  Call the Background task if no key is available}
                          {;}
                          {KeyStat:}
/$B4/$01
                     {    Mov    Ah,01            ;
                          Look for available key}
/$9C
                     {    Pushf                   ;
                          Call the keyboard function}
/$2E                {     CS:}
```

```
/$FF/$1E/>BIOS_INT16
                    {     Call dw [<BIOS_INT16]}
/$74/$01
                    {     Jz      ChkDosCr        ;
                     No Character available from Keyboard}
/$C3
                     {    RET                     ;
                     else return with new flags and code}
                          {ChkDosCr:}
/$06
                    {     Push    ES              ;
                  Check if DOS Critical error in effect}
/$56                      {       Push    Si}
/$2E                      {       CS:}
/$C4/$36/>DOSSTAT2
                   {      Les     Si,[>DOSStat2]}
/$26
                         {ES:
                      ; Zero says DOS is interruptable}
/$AC
              {           Lodsb                   ;
                    $FF  says Dos is in a critical state}
/$2E                      {       CS:}
/$C4/$36/>DOSSTAT1
                  {       Les     Si,[>DosStat1]  ;
                     If INDOS then INT $28 issued by DOS}
/$26
                     {    ES:                     ;
                          so we dont have to.}
/$0A/$04             {    Or      Al,[SI]}
/$2E
                     {    CS:                     ;
                        Account for active interrupts}
/$0A/$06/>INTR_FLAGS
                  {       Or      Al,[<Intr_Flags];
                       Any flags says we dont issue call}
/$5E
                {         Pop     Si              ;
                        to the background.}
/$07            {         Pop     Es}
/$3C/$01
                {         Cmp     Al,01           ;
                       Must be INDOS flag only}
/$7F/$02
                 {        JG      Skip28          ;
                 DOS cannot take an interrupt yet}
/$CD/$28
                {         INT     $28             ;
            Call Dos Idle function (background dispatch).}
                          {Skip28:}
```

```
  /$31/$C0
                {           Xor     Ax,Ax           ;
                          Show  no keycode available}
  /$C3                        {  RET  }
  {;-----------------------------------}
);


{*******************************
*  S T A Y I 1 3 . 4 1 0       *
*******************************}


Inline(
                                  {; STAYI13.400}
                                  {;-----------;}
  {; Routine to Set a Flag when INT 13 Disk I/O is active}
  $5D                         Pop   Bp ;
                              Remove Turbo stack frame}
  /$5D                        {         Pop   Bp}
  /$2E                        {         CS:}
  /$80/$0E/>INTR_FLAGS/<INT13_ON
                              {         Or by
      [<Intr_flags],<INT13_on      ; Say INT 13 is Active}
  /$9C
                {    Push     ; Invoke Original Disk INT 13}
  /$2E                        {         CS:}
  /$FF/$1E/>BIOS_INT13
                           {   Call dw [<BIOS_INT13]}
  /$9C
             {                Pushf              ;
                              Save Return Flags}
  /$2E                        {         CS:}
  /$80/$26/>INTR_FLAGS/<FOXS-INT13_ON
                  {     And by [<Intr_flags],<Foxs-INT13_on;
                              Clear INT 13 Active flag}
  /$9D
                              {         Popf       ;
                              Retrieve results flags}
  /$CA/$02/$00
                              {         RETf  2          ;
                              Throw away old flags}

{;.................................................}
);
```

```
{**********************************
*    S T A Y I 2 1 . 4 1 0       *
**********************************}


Inline(
                                   {; STAYI21.400}
                                   {;-----------}
{; Routine to Set a Flag when certain INT21 functions are
active. Functions to be flagged are identified in the main
Stayres routine. Cf. Functab array.}
  $5D
                      {    Pop   Bp ; Remove Turbo Prologue}
  /$5D
                           {         Pop   Bp}
  /$9C                     {         PushF}
  /$FB
                     {     STI                              ;
                           Allow interrupts}
  /$80/$FC/$62
              {          Cmp   Ah,$62                     ;
                        Verify Max function}
  /$7F/$28                 {         Jg    SkipI21}
{; Some Int 21 functions must be left alone. They either
never return, grab parameters from the stack, or can be
interrupted.   This   code   takes   account   of   those
possibilities.}
  /$50
                  {        Push  Ax                        ;
                       Skip functions marked 1 in}
  /$53
                  {        Push  Bx                        ;
                        in the function table.}
  /$86/$C4
                        {  Xchg  Ah,Al}
  /$BB/>FUNCTAB
                 {         Mov   Bx,>FuncTab               ;
                       Test Int 21 function}
  /$2E                 {         CS:}
  /$D7                 {         Xlat}
  /$08/$C0
                  {        Or    Al,Al                     ;
                   Wait for functions marked zero}
  /$5B
                   {       Pop   Bx                        ;
                        in the function table.}
  /$58                 {         Pop   Ax}
  /$75/$19             {         Jnz   SkipI21}
                       {SetI21:}
```

```
  /$2E                          {         CS:}
  /$80/$0E/>INTR_FLAGS/<INT21_ON
                {               Or by [<Intr_flags],<INT21_on ;
                                Say INT 21 is Active}
  /$9D                          {         PopF}
  /$9C                          {         Pushf}
  /$2E                          {         CS:}
  /$FF/$1E/>DOS_INT21
                {              Call dw [<DOS_INT21]
                               ; Invoke Original INT 21}
  /$FB
                {               STI
                    ; Insure interrupts enabled}
  /$9C
                 {              Pushf                              ;
                     Save Return Flags}
  /$2E
                                {         CS:    ;
                    Clear INT 21 Active}
  /$80/$26/>INTR_FLAGS/<FOXS-INT21_ON
          {             And by [<Intr_flags],<Foxs-INT21_on}
  /$9D
                  {             Popf                               ;
                     Retrieve the flags}
  /$CA/$02/$00                  {         RETF  2}
                                {SkipI21:
                      ; Invoke Int 21 w/o return}
  /$9D                          {         PopF}
  /$2E                          {         CS:}
  /$FF/$2E/>DOS_INT21
                  {              Jmp dw [>Dos_INT21]}

{;.................................................}
);

{*********************************************************}
{*   C L K I 8 . 4 1 0    Clock Interrupt Service        *}
{*********************************************************}
(* CLOCK_I8.INL *)
(*  Fm: Neil J. Rubenking [72267,1531]
     On each call to INT 8, this routine checks if the
timer is "running".  If it is, it checks if the activation
time has been reached.  If it has, the STATUS byte is set
to include the "HotKey_On" and "From_Timer" bits.  After
that, control  passes on to the STAYI8.OBJ code *)
  (*NJR*)
  INLINE(
  $9C/                          {PUSHF}
  $2E/$F6/$06/>Status/<Timer_On/
                    {TEST BY CS:status, timer_on}
  $74/$29/                      {JZ nothing}
```

```
  $50/                          {PUSH AX}
  $1E/                          {PUSH DS}
  $B8/$40/$00/                  {MOV AX,40h}
  $8E/$D8/                      {MOV DS,AX}
  $A1/$6E/$00/                  {MOV AX,[6E]}
  $2E/$39/$06/>timer_hi/        {CMP CS:timer_hi,AX}
  $75/$16/                      {JNZ not_yet}
  $A1/$6C/$00/                  {MOV AX,[6C]}
  $2E/$39/$06/>timer_Lo/        {CMP CS:timer_Lo,AX}
  $7D/$0C/                      {JGE Not_Yet}
  $2E/$80/$0E/>Status/<HotKey_On/
                                {OR BY CS:status, hotkey_on}
  $2E/$80/$0E/>Status/<from_Timer/
                               {OR BY CS:status, from_timer}
{Not_Yet}
  $1F/                          {POP DS}
  $58/                          {POP AX}
{nothing}
  $9D);                         {POPF}
  (*NJR*)
{---------- E n d  C l o c k _ I 8 --------------------}


{**********************************
*     S T A Y I 8 . 4 2 0        *
**********************************}

Inline(
                                    {; STAYI8.413}
                                    {;-----------}
{; Routine to Await Outstanding I/O, then post Stayres
Active}
  $5D
                                {    Pop   Bp   ;
                                Remove Turbo Prologue}
  /$5D                          {    Pop   Bp}
  /$9C                          {    Pushf}
  /$2E                          {    CS:}
  /$FF/$1E/>BIOS_INT8
                  {    Call dw [>BIOS_INT8]          ;
                      Invoke Original INT 8}
  /$2E                          {    CS:}
  /$F6/$06/>STATUS/<HOTKEY_ON
                       {       Test by [<Status],<HotKey_on  ;
                               Have we received the HOKEY}
  /$74/$39                      {    Jz    NoGo}
  /$2E                          {    CS:}
  /$F6/$06/>STATUS/<INUSE
                        {      Test by [<Status],<Inuse      ;
                           If Inuse.. then No go}
  /$75/$31                      {    Jnz   NoGo}
```

```
  /$2E
                              {    CS:              ;
                              Have the HotKey}
  /$80/$3E/>WAITCOUNT/$00
                         {    Cmp by [<WaitCount],00          ;
                           If waiting, check time}
  /$75/$22                    {    Jnz   Waiting}
{; If Not already waiting I/O, not already in use, and
HotKey received see if DOS is now interruptable}
                              {ChkIO:}
  /$06
                              {    Push  ES              ;
                             Save registers}
  /$56                        {    Push  Si}
  /$50                        {    Push  Ax}
  /$2E                        {    CS:}
  /$C4/$36/>DOSSTAT1
                        {      LES   Si,[>DOSstat1]            ;
                          Fetch Dos status 1}
  /$26                        {    ES:}
  /$AC
                          {   Lodsb                            ;
                          Fetch Status byte from dos}
  /$2E                        {    CS:}
  /$C4/$36/>DOSSTAT2
                        {     LES   SI,[>DOSstat2]            ;
                            Add second status byte}
  /$26                        {    ES:}
  /$0A/$04                    {    Or    Al,[SI]}
  /$2E                        {    CS:}
  /$0A/$06/>INTR_FLAGS
                        {     Or    Al,[<Intr_Flags]          ;
                         Add Interrupt active flags}
  /$58                        {    Pop   Ax}
  /$5E                        {    Pop   Si}
  /$07                        {    Pop   ES}
  /$74/$0E
                        {     Jz    Go                        ;
                          Wait for inactivity}
  /$2E                        {    CS:}
  /$C6/$06/>WAITCOUNT/$10
                        {     Mov by [<WaitCount],$10         ;
                              Set Wait count}
                              {Waiting:}
  /$2E                        {    CS:}
  /$FE/$0E/>WAITCOUNT
                       {    Dec by [<WaitCount]              ;
                          Decrement wait count}
  /$74/$D7                    {    Jz   ChkIO}
                              {NoGo:}
  /$CF                        {    IRET}
```

```
    {GO:    ; Enter the User's Turbo Procedure}
  /$2E                          {     CS:}
  /$FF/$16/>USERPROGRAM         {     Call [<UserProgram]}
  /$CF                          {     IRET}

{;.................................................}
);


{*********************************
*    S T A Y I 2 8 . 4 1 0      *
*********************************}

Inline(
                                    {; STAYI28.400}
                                    {;------------}
 {; Routine to Invoke User Code When HotKey or DOS idle}
  $5D
                                {          Pop   Bp    ;
                                Remove Turbo Prologue}
  /$5D                          {          Pop   Bp}
  /$9C                          {          Pushf}
  /$2E                          {          CS:}
  /$FF/$1E/>DOS_INT28
              {          Call dw [>DOS_INT28]
                       ; Invoke Original INT 28}
  /$2E                          {          CS:}
  /$F6/$06/>STATUS/<HOTKEY_ON
                 {          Test by [<Status],<HotKey_on  ;
                            Have we received the HOKEY}
  /$74/$25                      {          Jz    NoGo}
  /$2E                          {          CS:}
  /$F6/$06/>STATUS/<INUSE
                   {          Test by [<Status],<Inuse      ;
                         If Inuse.. then No go}
  /$75/$1D                      {          Jnz   NoGo}
{; If Not already waiting I/O, not already in use, and
HotKey received see if DOS is now interruptable}
                                {ChkIO:}
  /$06
                {          Push  ES                      ;
                         Save registers}
  /$56                          {          Push  Si}
  /$50                          {          Push  Ax}
  /$2E                          {          CS:}
  /$C4/$36/>DOSSTAT2
                   {          LES   SI,[>DOSstat2]          ;
                      Fetch DOS Critical status byte}
  /$26                          {          ES:}
  /$AC                          {          LodSb}
  /$2E                          {          CS:}
```

```
  /$0A/$06/>INTR_FLAGS
                    {       Or    Al,[<Intr_Flags]        ;
                        Add Interrupt active flags}
  /$58                      {         Pop   Ax}
  /$5E                      {         Pop   SI}
  /$07                      {         Pop   ES}
  /$75/$09
                {           Jnz   NoGo                    ;
                     Wait for inactivity}
  /$2E
                    {       CS:                           ;
                            Have the HotKey}
  /$80/$3E/>WAITCOUNT/$00
                   {        Cmp by [<WaitCount],00        ;
                            If timer waiting, go}
  /$E9/$01/$00              {         Jmp Go}
                            {NoGo:}
  /$CF                      {         IRET}
                            {GO:
                          ; Enter the User's Turbo Procedure}
  /$2E                      {         CS:}
  /$C6/$06/>WAITCOUNT/$00
                {           Mov by [<WaitCount],00        ;
                          Kill INT8 wait count}
  /$2E                      {         CS:}
  /$FF/$16/>USERPROGRAM
                    {       Call [<UserProgram]}
  /$CF                      {         IRET}
{;.....................................................}

Inline(

{;*******************************************************;}
{;    S  T  A  Y  S  A  V  E  .  4  2  0                ;}
{;*******************************************************;}
                            {;Version 4.15}
                                {;}
{; This Inline routine will save the regs and Stack for
Stay resident programs. It restores DS and SS from the
previously saved integer constants "OurDseg" and
"OurSSeg". DS is restored from the Turbo Initialization
Savearea.}
{; Author: Copyr. 1985, 1986}
{;                Lane Ferris}
{;              - The Hunter's Helper -}
{;Distributed to the Public Domain for use without profit.}
{;              Original Version 5.15.85}
```

```
  $FA
                              {       CLI  ;
                               Stop all interrupts}
  /$2E                        {       CS:}
  /$80/$0E/>STATUS/<INUSE
                              { Or by   [<Status],<InUse  ;
                            Set Active bit}
            {; Switch the SS:Sp reg pair over to ES:Si}
            {; Put Turbo's Stack pointers into SS:Sp}
  /$2E                        {       CS:}
  /$8C/$1E/>USRDSEG
                        {     Mov     [>UsrDSeg],DS       ;
                              Save Usr DataSegment}
  /$2E                        {       CS:}
  /$8C/$16/>USRSSEG
                      {       Mov     [>UsrSSeg],SS       ;
                              Save Usr Stack Segment}
  /$2E                        {       CS:}
  /$89/$26/>USRSPTR
                      {        Mov     [>UsrSPtr],Sp       ;
                               Save Usr Stack Ptr}
         {; Stack User interrupted pgm regs for Exit.}
        {; These are the original interrupt process regs}
          {; that must be returned on interrupt return}
   /$2E                       {       CS:}
  /$8E/$1E/>OURDSEG           {       Mov     DS,[>OurDseg] ;
 Get Turbo Stack pointer from DataSegment}
  /$2E                        {       CS:}
  /$8E/$16/>OURSSEG
                              {       Mov     SS,[>OurSSeg]}
  /$8B/$26/$74/$01
                              {       Mov     Sp,[$174]     ;
             Sp set by code at $B2B in Turbo initialization}
  /$55                        {       Push    Bp}
  /$50                        {       Push    Ax}
  /$53                        {       Push    Bx}
  /$51                        {       Push    Cx}
  /$52                        {       Push    Dx}
  /$56                        {       Push    Si}
  /$57                        {       Push    Di}
  /$06                        {       Push    Es}
    {; Save the InDOS stack to avoid recursion crashes
(Writeln).}
             {; Setup destination to Turbo Stack}
  /$89/$E7                    {       Mov     Di,Sp  ;
                               Dest is our stack}
  /$4F                        {       Dec     Di     ;
                               Back off current used word}
  /$4F                        {       Dec     Di}
  /$2E                        {       CS:}
  /$8C/$D0                    {       Mov     Ax,SS  ;
```

```
                                Turbo stack is destination}
  /$8E/$C0                     {       Mov     ES,Ax}
    {; Setup source from DOS Indos primary stack}
  /$2E                         {       CS:}
  /$8E/$1E/>DOSSSEG            {       Mov DS,[>DosSSeg]   ;
Source is DOS Indos primary stack}
  /$2E                         {       CS:}
  /$8B/$36/>DOSSPTR
                               {       Mov     Si,[>DosSptr]   ;
                                DOS primary stack offset}
  /$B9/$40/$00                 {       Mov     Cx,$40}
  /$2E                         {       CS:}
  /$89/$0E/>DOSSSIZ            {       Mov     [>DosSsiz];
                                remember the stack word size}
  /$4E                         {       Dec     Si      ;
                                    point last word on stack}
  /$4E                         {       Dec     Si}
  /$89/$E0
                               {       Mov     Ax,Sp           ;
                              Get stack pointer higher to avoid}
  /$29/$C8
                               {       Sub     Ax,Cx
                 ;overwriting during enabled REP functions}
  /$29/$C8                     {       Sub     Ax,Cx}
  /$89/$C4                     {       Mov     Sp,Ax}
  /$FD                         {       STD             ;
                                Move like Pushes on stack}
  /$F2/$A5
                               {       Rep Movsw       ;
                                  Move users stack to our own}
  /$89/$FC
                               {       Mov     Sp,Di   ;
            Update our stack pointer to available word.}
  /$FC                         {       Cld}
  /$2E                         {       CS:}
  /$8E/$1E/>OURDSEG
                       {       Mov     DS,[>OurDSeg]   ;
                         Setup Turbo Data Segment Pointer}
  /$FB
                               {       STI             ;
                                  Enable Interrupts}

{;.....................................................}
);
```

```
Inline(

{;*****************************************************;}
{  S  T  A  Y  R  S  T  R  .  4  2  0                  ;}
{  This is the StayRstr.Inc file included above        ;}
{;*****************************************************;}
                                  {;Version 4.15}
{ Inline Code to restore the stack and regs moved; to the
Turbo Resident Stack which allows Turbo Terminate & Stay
Resident programs.}
{       ; Copr. 1985, 1986}
{       ; Author: Lane Ferris}
{       ;       - The Hunter's Helper -}
{ Distributed to the Public Domain for use without profit.}
          {       ; Original Version 5.15.85}
{;-----------------------------------------------------;}
{; Restore the Dos (or interrupted pgm) Regs and Stack  ;}
{;-----------------------------------------------------;}
          {; Replace the Users Saved Stack}
{; Note that pushes on the stack go in the opposite
direction of our moves. Thus we dont worry about REP stack
activity overlaying the enabled REP fuction.}
  $FA                         {    CLI}
  /$2E
                              {    CS:     ;
             Avoid stack manipulation if never "StaySaved"}
  /$A1/>DOSSSIZ               {    Mov  Ax,[>DosSsiz]}
  /$09/$C0                    {    Or   Ax,Ax}
  /$74/$20                    {    Jz   NotinDos}
  /$8C/$D0
                              {    Mov  Ax,SS         ;
                              Source is our Stack}
  /$8E/$D8                    {    Mov  DS,Ax}
  /$89/$E6
                              {    Mov  Si,Sp         ;
                  Point to Last used USER word on our stack}
  /$46                        {    Inc  Si}
  /$46                        {    Inc  Si}
  /$2E                        {    CS:}
  /$8E/$06/>DOSSSEG
                              {    Mov  ES,[>DosSSeg] ;
                       Dest is Dos indos primary Stack}
  /$2E                        {    CS:}
  /$8B/$3E/>DOSSPTR           {    Mov  Di,[>DosSptr]}
  /$2E                        {    CS:}
  /$8B/$0E/>DOSSSIZ
                              {    Mov  Cx,[>DosSsiz] ;
                              Saved words}
  /$29/$CF
                              {    Sub  Di,Cx         ;
```

```
                                point to last used word of Dos stack}
        /$29/$CF                    {    Sub  Di,Cx}
        /$FC                        {    CLD}
        /$F2/$A5
                                {    Rep Movsw          ;Careful!
                                  Interrupt are enabled here}
        /$89/$F4
                                    {    Mov  Sp,Si         ;
                                    Skip over moved words}
                                    {;--}
                                    {NotinDos:}
        /$07                        {    Pop  Es}
        /$5F                        {    Pop  Di}
        /$5E                        {    Pop  Si}
        /$5A                        {    Pop  Dx}
        /$59                        {    Pop  Cx}
        /$5B                        {    Pop  Bx}
        /$58                        {    Pop  Ax}
        /$2E                        {    CS:}
        /$80/$26/>STATUS/<FOXS-INUSE-HOTKEY_ON
                  {  And by [<Status],<Foxs-Inuse-HotKey_on   ;
                                Clear INUSE flag}
        /$2E                        {    CS:;
                                      .. and HotKey}
        /$8E/$1E/>USRDSEG           {    Mov  DS,[<UsrDSeg]}
        /$2E                        {    CS:}
        /$8E/$16/>USRSSEG           {    Mov  SS,[<UsrSSeg]}
        /$2E                        {    CS:}
        /$8B/$26/>USRSPTR           {    Mov  SP,[<UsrSPtr]}
        /$5D
                                    {    Pop  Bp            ;
                                 Remove Bp,Sp from Procedure entry}
        /$5D                        {    Pop  Bp}
        /$FB
                                    {    STI                ;
                                      enable interrupts}
        /$C3                        {    RET}

    {;.....................................................}
```

REFERENCES

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

3. *Text-to-Speech Interface for the IBM Voice Communications Adapter, Talbot, Summer 1987*

PART II

VOICE RECOGNITION

and

APPLICATIONS

IBM Voice-Activated
Keyboard Utility
User Guide

Summer 1987

Gary L. Talbot

Management Information Systems Department

University of Arizona

Tucson, Arizona

TABLE OF CONTENTS

INTRODUCTION

The **IBM Voice-Activated Keyboard Utility** lets a user speak DOS commands or the commands that run application programs on a personal computer. The user talks into a microphone or a telephone attached to the computer through the **IBM Voice Communications Adapter** (a hardware board) and commands that have been trained to the individual's voice are executed. The action is like typing the actual commands on the computer's keyboard. This utility may be used transparently within any application.

As an example, once the user-defined vocabulary has been trained to a person's voice, they may speak "**directory alpha enter**" into the microphone or telephone and the directory command on the a: (alpha) drive will be executed. The command words are arranged such that only certain words are active at a particular time. That is, many of the DOS commands could be active initially but once one is voiced then only certain other words become active. For example, once the directory command has been voiced, then only the parameters wide, pause, enter, or cancel become active. This simulates the same order that is common to DOS commands entered through the keyboard.

In order to use a command vocabulary, each word must be trained to the individual user's voice. Words must also be trained to each individual user's voice in different environments. That is, if the vocabulary is trained by one user and a second person wishes to use the utility, then the second person has to retrain the words to their unique voice patterns. Also, if the original person trains the words with a quiet background and moves to a more noisy background, then the vocabulary may have to be retrained for full voice recognition. Care should be taken when training the words in the vocabulary since it is the most important factor in recognition accuracy.

This guide is intended to simplify the task of a user wishing to use the **IBM Voice-Activated Keyboard Utility** to voice their commands to the computer. The sections following will discuss hardware and software requirements that are necessary to use this program. Also, installation instructions for using this utility will be discussed. An overview of using the utility is covered under the operating instructions section. Next, input and output formats and descriptions are discussed. Finally, references for further investigation are provided.

HARDWARE AND SOFTWARE REQUIREMENTS

The **IBM Voice-Activated Keyboard Utility** works in conjunction with and through the **IBM Voice Communications Operating System** software and the **IBM Voice Communications Adapter** hardware board. Therefore, the user must ensure that both the software and hardware are installed on the computer at which they are going to use the **IBM Voice-Activated Keyboard Utility**. A microphone or telephone that is attached to the **IBM Voice Communications Adapter** to communicate to the computer is also required.

Hardware:

Minimum hardware requirements are given in Table 5.1.

MINIMUM HARDWARE REQUIREMENTS

- IBM PC/AT/XT or compatibles
- 160 KB memory
- Two double-sided diskette drives (360 KB / 1.2 MB) or one double-sided diskette drive (360 KB / 1.2 MB) and one fixed disk
- Monochrome or color monitor
- An IBM Voice Communications Adapter
- A high impededance microphone with an attached subminiature 2.5 mm (0.1 inch) connector or an FCC approved telephone set

Table 5.1

Software:

Minimum software requirements are given in Table 5.2.

MINIMUM SOFTWARE REQUIREMENTS

- DOS 2.10 or higher for IBM PC/AT or DOS 3.00 or higher for IBM XT
- IBM Voice Communications Operating Subsystem Program
- IBM Voice-Activated Keyboard Utility, 6489831

Table 5.2

INSTALLATION INSTRUCTIONS

Installation instructions for the **IBM Voice Communications Adapter** may be found in *IBM Installation and Setup Voice Communications, 6280711.* Basic installation can be accomplished in 30 minutes or less by an inexperienced person.

Installation instructions for the **IBM Voice Communications Application Program Interface** (the software driver) may be found in *IBM Voice Communications Application Program Interface Reference Vol 1 Chap 2, 6280743.* The software resides in a subdirectory, either on a hard drive or floppy diskette named *vcapi*. The Voice

Communications Operating Subsystem Program diskette is self installing and is a fairly simple procedure. Different procedures exist for installing the system on hard or floppy disks.

To load the operating system and the required discrete utterance recognition software, the following commands should be placed in the **autoexec.bat** file:

set vcapi = y:\vcapi

(where y is drive containing the vcapi directory and vcapi is the name of the DOS directory containing the API code.)

y:\vcapi\vcapidrv /o 11

(the /o 11 option allows the discrete utterance recognition function to be loaded when the API driver, vcapidrv, is loaded at boot time.)

To setup the **IBM Voice-Activated Keyboard Utility**, place the utility diskette in drive A and enter **keysetup**. A series of questions appear which should be answered according to the particular user's configuration. More specific installation procedures can be found in the *IBM Voice-Activated Keyboard Utility, 6489838, Chap 2.*

The **IBM Voice-Activated Keyboard Utility** is based on overlays, executable code segments that are loaded into memory only when needed. Once the operating system has been installed, the particular overlay that is desired to be used must be loaded. For example, each particular application may have a specific overlay that is setup for that application only.

Each overlay when developed is entered into a plain text file with the extension **.lan** according to the rules found in the above reference. It is then compiled using the **Voice Command Language Compiler (VOCL)** to create an executable overlay or a language description file with the extension **.ldf**.

OPERATING INSTRUCTIONS

To start the utility and load the console and DOS overlays, enter the batch file name **keyinit.bat**. This batch file executes a sequence of initialization commands, ie., sets up the specified path, turns the microphone or telephone on, installs the selected overlay, etc. Initial training must be accomplished at this point. An excellent tutorial is given in Chap 3 of reference 3 found at the end of this guide. Once the initial training is accomplished, then the vocabulary remains trained for the next and succeeding user sessions.

After the commands have been trained, they are now ready for use. If at any time, the user desires to see the available commands, they may voice the command "voice-menu" or enter ALT-M. A list of active words appear. Trained words are marked with an asterisk prefix and may be used at any time. Other keystrokes can also be set to activate special commands. Table 5.3 lists common VCOM command keys.

VCOM COMMAND KEYS

Keys	Voice Command	Associated VCOM command
ALT-C	Voice Console	vcom console
ALT-M	menu	vcom menu/permanent
ALT-L		vcom microphone on
ALT-O		vcom microphone off
ALT-T		vcom microphone momentary
ALT-R		vcom remember
ALT-D		vcom define

Table 5.3

ALT-C activates the voice console and allows several actions to be performed on the vocabulary such as defining words, training words, etc.

ALT-M displays the menu of current active words.

ALT-L turns the microphone on.

ALT-O turns the microphone off.

ALT-T turns the microphone on momentarily until someone speaks at which time it's turned off again.

ALT-R starts the remembering of a sequence of keystrokes.

ALT-D stops the remembering of a sequence of keystrokes and works in conjunction with the ALT-R command.

VCOM commands are commands that may be entered from the DOS command line and which are then passed via the program **vcom.com** to the utility. The **ALT** key combinations are shortcut ways to execute these same commands. These **VCOM** commands and others may be entered at any time from the DOS prompt.

Other overlays that have been created may be loaded using the VCOM command:

vcom overlay *filename*

The special overlay, ***console.ldf***, is included within the utility that allows the user to speak the **Voice Console** commands. This overlay may be loaded to remain resident, trained, then another user overlay may be loaded. The **console.ldf** remains activated so the user can speak the **Voice Console** commands such as "yes" or "no" and other training commands when using the second overlay.

In summary, the **IBM Voice-Activated Utility** runs in the background using trained voice commands to generate preprogrammed keystrokes that DOS or an application needs. Each application may have its own unique program or overlay. A special overlay, **console.ldf**, allows the user a way to voice commands to the **Voice Console**.

Training the vocabulary is one of the most critical aspects of using the **Voice-Activated Keyboard Utility**. Training options allow words to be trained in any order, varying the number of training instances (up to nine different samplings or instances of a word may be remembered; as the number of instances increases, the quality of voice recognition increases), and listing all words associated with a particular overlay. A speech test procedure is provided to assure that trained words are recognized and produce the desired keystrokes.

Each application's grammar may vary. For example, in DOS, only certain parameters can be input after one command is given as in the command "directory" followed by the parameter "wide". The user can establish their choice of active words whose keystrokes can be redefined at any time using the Voice Console. Different word groups can be activated when they are needed such as when the word "macro" is voiced. This command will activate an entire new set of voice selectable commands that have been previously defined. Finally, each user may establish and load unique overlays designed for their application.

The **VCOM.COM** program passes utility keyboard commands (VCOMs) entered at the DOS command prompt to the utility program. Various commands exist to do such things as loading and unloading overlays, selecting input device (microphone or telephone), selecting the number of training instances, turning the microphone/telephone on or off, etc. An example follows:

A>vcom overlay *dos*

which loads and activates the overlay ***dos.ldf*** into memory. **VCOM** commands available for use may be found in Chapter 5 of the *IBM Voice-Activated Keyboard Utility* manual.

A user may design an overlay specific to their application. An overlay tells the **Voice-Activated Keyboard Utility** the names of the words that can be spoken, the order in which the words can be spoken, the keystroke sequences generated by the words, the name of the word groups, and a list of commands (**VCOMs**) to be executed when the overlay is loaded. The following steps should be taken to create an overlay:

1. Create a text or language file (.lan) that defines the overlay.

2. Run the VOCL compiler to create an executable language definition file (.ldf).

3. Train the vocabulary using the Voice Console.

4. Test the overlay by using the speech test.

Specific rules and syntax for generating an overlay can be found in Chapter 7 of the *IBM Voice-Activated Keyboard Utility* manual.

An example of a simple user developed overlay follows:

```
menu "'a-m'";
voice_console "'a-c'";
line_up "'esc'H";
line_down "'esc'P";
scroll_up "'c-W'";
scroll_down "'c-Z'";
page_up "'esc'I";
page_down "'esc'Q";
delete_line "'c-Y'";
delete_character "'esc'S";
begin_block "'c-K'b";
end_block "'c-K'k";
copy_block "'c-K'c";
move_block "'c-K'v";
hide_block "'c-K'h";
delete_block "'c-K'y";
read_block "'c-K'r";
write_block "'c-K'w";
end "'c-K'd";
top_of_file "'c-Q'r";
end_of_file "'c-Q'c";
left "'esc'K";
right "'esc'M";
word_left "'c-A'";
word_right "'c-F'";
beginning_of_line "'esc'G";
end_of_line "'esc'O";
find "'c-Q'f";
replace "'c-Q'a";
quit "q";
edit "e";
compile "c";
options "o";
run "r";
save "s";
escape "'esc'";

/*  The root sentence definition follows  */

Root [enter menu voice_console cmd1 cmd2 cmd3 cmd4 cmd5
      cmd6 cmd7 cmd8 cmd9 cmd10] =

(line_up,
 line_down,
 scroll_up,
 scroll_down,
 page_up,
 page_down,
 delete_line,
 delete_character,
 begin_block,
```

REFERENCES

1. *IBM Installation and Setup Voice Communications, 6280711*

2. *IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743*

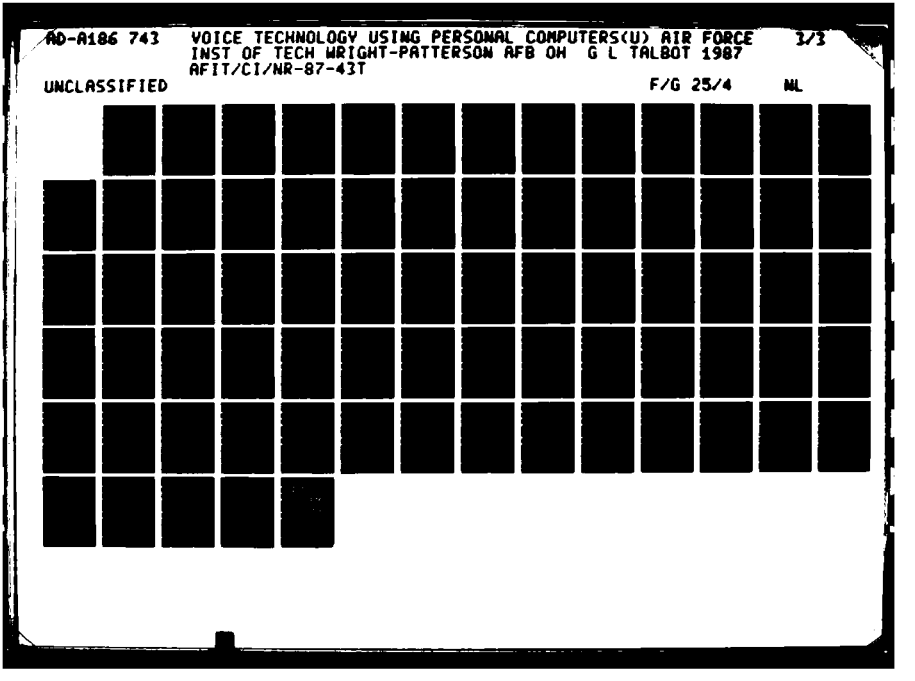
3. *IBM Voice-Activated Keyboard Utility, 6489838*

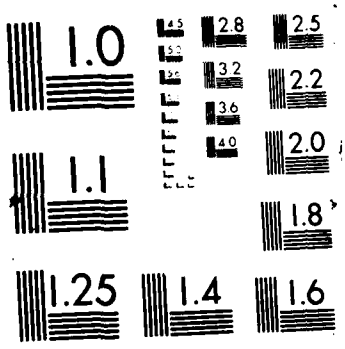
BIBLIOGRAPHY

IBM Installation and Setup Voice Communications, 6280711

IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743

IBM Voice-Activated Keyboard Utility, 6489838


AD-A186 743 VOICE TECHNOLOGY USING PERSONAL COMPUTERS(U) AIR FORCE 1/3
INST OF TECH WRIGHT-PATTERSON AFB OH G L TALBOT 1987
AFIT/CI/NR-87-43T
UNCLASSIFIED F/G 25/4 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

APPENDICES

of

PROGRAM LISTINGS

{SPEAK.INC}

```
{************************************************************
```

Procedure Speak serves as an interface to the IBM Voice Communications Applications Program Interface for the Text-to-Speech (speech synthesis) function set. It can be included in any Turbo Pascal program in which the user wishes to have a passage of text spoken. The only required lines within the calling program are a type declaration, an include statement to include the procedure, and the call to the procedure. Parameters that must be passed to the speak procedure are the name of a string containing a sentence of up to 240 ASCII text characters that ends with a sentence terminator('.','?',or '!'), an integer, p, giving the baseline pitch (the range for pitch is 0 or between 50 and 100), and an integer, r, which sets the speech rate (the range for speech rate, r, is between 50 and 250). A pitch of 0 will produce a whispering voice while other values not between 50 and 200 will default to the normal pitch rate of 85. Pitch may be adjusted at any time by replacing this value and speach will remain at this same pitch until another value is input. Resolution of baseline pitch is about 10 so differences such as 103 and 111 may not be detectable. Higher numbers produce higher pitches. If a value outside the range of 50 and 100 is not used with the speech rate, r, then the default is to the normal rate of 150 words per minute. Again, maximum resolution is about 10 words per minute so values such as 123 and 127 may not be detected. Speech rate is also adjustable by changing the value passed and this rate remains in effect until a different value is supplied. Higher values produce faster rates of speech.

When using Turbo Pascal, the compiler option {$V-} may be used to relax checking of the length of the buffer passed to the speak procedure. That is, a buffer with length of 80, 128, etc may be passed. However, it is still required to define a string of type 'longstr' for the var parameter used in the speak procedure.

An example user program follows:

```
************************************************************

Program Calling_Program;

{$V-}                {optional compiler directive to relax
                      parameter length checking}
```

```
type   longstr = string[240];   {this must be supplied
                                 since it is declared
                                 as a Var parameter in
                                 Speak}

     {other user declarations, variables,
             functions, procedures, etc.}


{$I speak.inc}     {includes the speak interface code}

begin              {user program starts here}

{user code to set up a buffer to send to speak
   procedure}

speak(string,p,r)   {call the speak procedure to speak
                       text in the string at pitch, p,
                       and at rate, r}

{more user code if desired}

end.                  {end of user program}

*******************************************************
Reference:  IBM Voice Communications Application
             Program Interface Reference   Vol 1 & 2

 For additional information on error codes returned,
 see the above reference or consult the Text-to-Speech
 Interface for the IBM Voice Communication Adapter
 Guide,  Talbot, Summer, 1987.

*******************************************************}


Procedure Speak(Var talk:longstr; p,r: integer);

label loop,99,fini;

type  result   = record
                   ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
                 end;

      plist    = array[0..5] of integer;

      shortstr = string[16];
```

```
var   reg    :result;          {record type to call
                                interrupt}
      rcb    :integer;         {storage for the resource
                                control block}
      bid    :integer;         {storage for the base id}
      cid    :integer;         {storage for partition 2
                                connection id}
      pb     :plist;           {the parameter block}
      k      :integer;         {length of text}
      pitch  :string[3];       {voice pitch string}
      rate   :string[3];       {voice rate string}
      setbuf :shortstr;        {set pitch and rate
                                buffer}

begin

{setup pitch, p, and speech rate, r}

if p in [0,50..200]
   then str(p,pitch)           {convert pitch to string}
   else pitch:= '85';          {default to normal pitch}

if r in [50,250]
   then str(r,rate)            {convert rate to string}
   else rate:= '150';          {default to normal rate}

setbuf:=^[+'['+pitch+'p'+^[+'['+rate+'r'+^@;
                    {setup the pitch and rate buffer}


{open command to obtain a resource control block and
 connection ids}

reg.ax:=$1111;                 {function code for open
                                command}
reg.dx:=$021f;                 {board I/O address}
reg.es:=seg(pb);               {parameter block segment}
reg.bx:=ofs(pb);               {parameter block offset}
intr($14,reg);                 {call interrupt 14}
if pb[0] <> 0 then             {zero if no error}
   begin
     writeln('An error occurred in open.');
     goto 99;
   end;
rcb:=pb[1];                    {save resource control
                                block}
bid:=pb[2];                    {save base id}
cid:=pb[4];                    {save partition 2
                                connection id}
```

```
{claim h/w resources for the rcb using claimhdw
   command}

reg.ax:=$111a;                  {function code for
                                 claimhdw command}
reg.dx:=bid;                    {need base id in dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[2]:=$2602;                   {claim port 2, partition
                                 2, spkr, microphone}
pb[3]:=$0000;                   {no base interrupt}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in claim.');
     goto 99;
   end;

{connect devices to the port using conndtop command}

reg.ax:=$1121;                  {function code for
                                 conndtop}
reg.dx:=bid;                    {need base id in dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[2]:=2;                       {connect to port 2}
pb[3]:=$0600;                   {connect microphone and
                                 speaker}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in connect devices.');
     goto 99;
   end;

{load function set into a port and connect it using
 the connftop command}

reg.ax:=$111f;                  {function code for
                                 connftop command}
reg.dx:=bid;                    {need base id in dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[1]:=cid;                     {need cid in the
                                 parameter block}
pb[2]:=2;                       {connect to port 2}
pb[3]:=10;                      {connect text-to-speech
                                 function}
```

```
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in connect
              function.');
     goto 99;
   end;

{the initialize text-to-speech function set data
   structures}

reg.ax:=$1113;                  {function code for
                                 initialize data
                                 structures}
reg.dx:=cid;                    {need connection id in
                                 dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[1]:=cid;                     {need cid in parameter
                                 block also}
intr($14,reg);                  {call interrupt 14}
if pb[0] <> 0 then              {zero if no error}
   begin
     writeln('An error occurred in initialize speech
              function.');
     goto 99;
   end;

{the text-to-speech speak command}

{set the pitch and rate by outputting setbuf}

reg.ax:=$111e;                  {function code for speak
                                 command}
reg.dx:=cid;                    {need connection id in
                                 dx}
reg.es:=seg(pb);                {parameter block segment}
reg.bx:=ofs(pb);                {parameter block offset}
pb[1]:=cid;                     {need cid in parameter
                                 block also}
pb[2]:=2;                       {32 bit address for
                                 buffer setbuf}
pb[3]:=ofs(setbuf)+1;           {setbuf address offset,
                                 offset 1 for length}
pb[4]:=seg(setbuf);             {setbuf address segment}
intr($14,reg);                  {call interrupt 14}
```

```
if pb[0] <> 0                    {zero if no error}
   then
      begin
        writeln('An error occurred in speech
                 function.');
        goto 99;
      end;

{say the text line that was passed as a parameter}

reg.ax:=$111e;                   {function code for speak
                                  command}
reg.dx:=cid;                     {need connection id in
                                  dx}
reg.es:=seg(pb);                 {parameter block segment}
reg.bx:=ofs(pb);                 {parameter block offset}
pb[1]:=cid;                      {need cid in parameter
                                  block also}
pb[2]:=2;                        {32 bit address for
                                  buffer talk}
k:=length(talk);                 {find the length of the
                                  buffer}
talk[k+1]:=^[;                   {put in an ESC}
talk[k+2]:='[';                  {and a left bracket}
talk[k+3]:='i';                  {and an i to create
                                  interrupt}
talk[k+4]:=^@;                   {add a null at the end}
pb[3]:=ofs(talk)+1;              {use the buffer passed in
                                  talk, offset 1 for
                                  length}
pb[4]:=seg(talk);                {segment for talk}
intr($14,reg);                   {call interrupt 14}
if pb[0] <> 0                    {zero if no error}
   then
      begin
        writeln('An error occurred in speech
                 function.');
        goto 99;
      end;
goto fini;

99:  writeln('Return Code is ',pb[0]);  {tell the user
                                         what code was
                                         returned}
```

```
{close command to release resources}

fini:                              {come here always to
                                    release resources}
reg.ax:=$1112;                     {function code for close}
reg.dx:=bid;                       {need base id in dx}
reg.es:=seg(pb);                   {parameter block segment}
reg.bx:=ofs(pb);                   {parameter block offset}
pb[1]:=rcb;                        {resource control block
                                    to release resources}
intr($14,reg);                     {call interrupt 14}
if pb[0] <> 0 then                 {zero if no error}
   begin
     writeln('An error occurred in close.');
     writeln('Return Code is ',pb[0]);  {tell the user
                                         what code was
                                         returned}
   end;

end;                               {procedure speak}
```

APPENDIX B

PROGRAM LISTING

for

SAY.COM

{SAY.PAS}

```
{This program will say the text entered as parameters on
the command line. Input is limited only to 127 total
characters (due to limitation of Turbo Pascal). To use the
program, enter the command 'say' followed by the text you
wish spoken. Remember to end the text with a sentence
terminator, either a period(.), question mark(?), or an
exclamation point(!). Examples:

    say This is a mighty fine computer!

    say Do you want to delete all files?

    say It is now time to have a coffee break.

{$V-}                                   {compiler directive to
relax
                                        length of strings}

type   longstr = string[240];     {size of buffer for text
input}

       word    = string[80];      {size of buffer for a word
                                   input}

var    passage  :longstr;    {buffer for text that is input}
       param    :word;       {buffer for word that is input}
       numparam :integer;    {number of parameters (words)}
       i        :integer;    {an index for words}

{$I b:speak.inc}                        {interface procedure for
                                         speech}
begin
  fillchar(passage,240,' '); {clear the buffer}
  numparam:=paramcount;      {find number of words passed}
  for i:= 1 to numparam do   {create the text buffer}
    begin                    {to be spoken}
      param:=paramstr(i);    {get each word from the command
                              line}
      passage:=passage+' '+param;  {and add it to the text
                                    buffer}
    end;                     {end for i:=1 to numparam}
  i:=length(passage);        {find the text length}
  if not (passage[i] in ['.','?','!']) then
     passage:=passage+'.';   {default to period if not
                              punctuated}
  speak(passage,65,170);     {speak the text in buffer,
                              pitch 65, rate 170}
end.
```

APPENDIX C

PROGRAM LISTING

for

SAYTEXT.COM

{SAYTEXT.PAS}

```
{************************************************************
```
The SAYTEXT program causes the text in the file, whose name is passed as a parameter, to be spoken. The file should be in regular ASCII characters similiar to this passage following all rules of normal punctuation. The length of the input file is unlimited. The text is first read into a linked list then each node of the linked list is spoken.

Example:

if the file named HELLO.DAT contains the following text:

> Hello all! It is so nice of you to visit. Will you come in and stay for awhile?

then to have the passage spoken, enter the following command:

```
      saytext hello.dat
************************************************************}
program saytext;

{$V-}  {compiler directive to relax length of parameter
        strings passed}

type   longstr = string[240];     {length of text string to
                                   speak}
       filname = string[66];      {file name passed}
       buffer  = array[1..240] of char; {temporary buffer
                                         storage}

{******* RECORD FOR LINKED LIST NODE ***********}
    LlistNod = ^SNode;
    SNode = record
            txt: longstr;
            next: LlistNod;
            prior: LlistNod;
    end;

{******* RECORD FOR LINKED LIST HEADER ***********}
    Slist = ^SHead;
    SHead = record
            length: integer;
            first:  LlistNod;
            Last:   LlistNod;
    end;
```

```
var     data      :filname;       {buffer to hold file name
                                   that is passed}
        datafile  :text;          {assigned to the filename}
        i,j       :integer;       {counter for nodes and
                                   chars}
        LList     :Slist;         {the head node}
        node      :LListnod;      {pointer to keep track of
                                   current node}
        Str240    :longstr;       {buffer for string}
        chin      :char;          {char read in}
        buf       :buffer;        {used to manipulate data}

{*********************************************************}
{*      function to test for existence of a file          *}
{*********************************************************}
Function Exist(filename: filname): boolean;

var     fil       :file;

begin
  assign(fil,filename);
  {$I-}
  reset(fil);
  {$I+}
  exist:= (IOresult = 0)
end;{function exist}

{*************************************************************
*Node_Ptr; RETURNS A PTR  TO  CURRENT NODE OF LINKED LIST
*************************************************************}

Function Node_Ptr(pos: integer): LlistNod;

Var
   i:  integer;
   nd: LlistNod;

Begin
  nd := Llist^.first;
  for i := 2 to pos do
    nd := nd^.next;
  Node_Ptr := nd;
End;
```

```
{************************************************************
* CreateLst; CREATES HEADER FOR LINKED LIST FOR TEXT LINES*
************************************************************}

Function CreateLst: Slist;

Var
   thishead: Slist;

Begin
  new(thishead);
  thishead^.length := 0;
  thishead^.first := nil;
  thishead^.last := nil;
  CreateLst := thishead;
End;

{************************************************************
* Make_Node;CREATES NEW NODE FOR LINKED LIST                *
************************************************************}

Function  Make_Node(dat:  longstr;  prev,  nxt:  LlistNod):
LlistNod;

Var
   thisone: LlistNod;

Begin
  new(thisone);
  thisone^.txt := Copy(dat,1,Length(dat));
  thisone^.prior := prev;
  thisone^.next := nxt;
  Make_Node := thisone;
End;

{************************************************************
* APP_Llist; APPENDS A NODE ONTO LINKED LIST                *
************************************************************}
Procedure App_Llist(dat: longstr);

Var
   thisone: LlistNod;
```

```
Begin
  if Llist^.first = nil then
    begin
      thisone := Make_Node(dat,nil,nil);
      Llist^.last := thisone;
      Llist^.first := thisone;
    end
  else
    begin
      thisone := Make_Node(dat,Llist^.last,nil);
      Llist^.last^.next := thisone;
      Llist^.last := thisone;
    end;
  Llist^.length := Llist^.length + 1;
End;

{************************************************************
*  DelHere;  DELETES A NODE FROM THE TEXT LINKED LIST AND *
*        RETURNS THE TEXT STRING FROM THAT NODE            *
************************************************************}
Function DelHere(pos: integer): longstr;

Var
   temp: LListNod;

Begin
  temp := Llist^.first;
  if pos = 1 then
    begin
      Llist^.first := temp^.next;
      if Llist^.first <> nil then
        Llist^.first^.prior := nil;
    end
  else
    begin
      temp := Node_Ptr(pos);
      temp^.prior^.next := temp^.next;
      if temp^.next = nil then
        Llist^.last := temp^.prior
      else
        temp^.next^.prior := temp^.prior;
    end;
  DelHere := temp^.txt;
  Dispose(temp);
  Llist^.length := Llist^.length - 1;
End;
```

```
{*************************************************************
*   DEALL_LIST;                                               *
*************************************************************}
Procedure Deall_List;

Var
  Tx : String[80];

Begin
  while Llist^.length > 0 do
    Tx := DelHere(1);
  Dispose(Llist);
End;

{$I b:speak.inc}                    {speech interface procedure}

begin
  data:=paramstr(1);                {get the file name passed as
                                     a parameter}
  if exist(data) then               {see if the filename is
                                     valid}
    begin                           {do this if filename valid
                                     else tell user}
      assign(datafile,data);        {assign var datafile to the
                                     string name}
      reset(datafile);              {get the file ready to read}
      LList:=CreateLst;             {create a head node}
      while not eof(datafile) do
        begin                       {begin while not eof...}
          j:=1;                     {initialize char counter}
          repeat
            read(datafile,chin);        {read char in}
            buf[j]:=chin;               {put it in an array}
            j:=j+1;                     {increment the index}
          until (chin in ['.','?','!']) or (j > 240) or
                eof(datafile);      {stop for end of sentence or
                                     buffer full or end of file}
          Str240:=copy(buf,1,j-1);  {creates a complete
                                     sentence}
          APP_LList(Str240);        {add it to the array}
        end;                        {while not eof(datafile)}
      close(datafile);              {remember to close the file}
      node:=LList^.first;           {set pointer to first node}
      for i:=1 to LList^.length do
        begin                       {for i:=1 to LList...}
          speak(node^.txt,65,175); {speak the current line}
          node:=node^.next;            {move the pointer up}
        end;                        {for i:=1 to LList...}
      Deall_List;                       {delete all the nodes}
    end                             {while not eof...}
```

```
    else
      writeln(data,' does not exist.');
                                {error message if file does not
                                 exist}
end.                            {program SAYTEXT}
```

APPENDIX D

PROGRAM LISTING

for

REMIND.COM

```
{REMIND.PAS}

{$R+}
{$C-}
{$V-}

PROGRAM REMIND;
{This program is a memory resident program that drives the
IBM voice applications software and hardware board.  When
first loaded, it allows input of messages to be spoken and
the time when they should be spoken. Interface to the board
is made through the procedure speak.  After the user is
prompted for input (messages and times), the program
terminates and becomes memory resident.  Access to the
schedule for review/revision or to terminate the program
can be made by entering ALT-F7.}


{ * * * * * CONSTANTS * * * * * * * * * * * * * * * * * * * * }
  const
{the next field is needed for the windo.inc routines }
    MaxMsg           = 15;  {maximum number of messages to
                               be in schedule}
    MaxWin           = 10;  {Max number of windows open at
                               one time }
    Esc              = #27; {character equivalent of Escape
                                 Key}
    Alt              = 08;  {Shift bits at 40:17 }
    Ctrl             = 04;
    Left_Shift       = 02;
    Rght_Shift       = 01;

    BIOSI8           = 8;   {Bios Timer interrupt}
    BIOSI16          = $16; {Bios Keyboard interrupt}
    BIOSI13          = $13; {Bios Disk interrupt}
    DOSI21           = $21; {DOS service router interrupt}
    DOSI28           = $28; {DOS Idle interrupt}
{------------- T Y P E    D E C L A R A T I O N S -------}
  Type
    Regtype      = record
                    Ax,Bx,Cx,Dx,Bp,Si,Di,Ds,Es,Flags:integer
                   end;

    HalfRegtype = record
                    Al,Ah,Bl,Bh,Cl,Ch,Dl,Dh:byte
                   end;

    filename_type = string[64];
```

```
    Vector    = record       { Interrupt Vector type    }
                  IP,CS :integer ;
                end ;
    longstr = string[24];
{-------------- T Y P E D   C O N S T A N T S ----------}
  Const

    Our_HotKey    : byte = 110;    { scan code for ALT-F7}

{****** scan code can be changed to make *****************}
{****** another key active as the hot key. ***************}

      { This table marks those INT 21 functions which must
be passed without modification. They either never return,
fetch parameters from the stack, or may be interrupted by a
TSR }

      Functab     : array[0..$6F] of byte =
              (1,1,1,1, 1,1,1,1, 1,1,1,1, 1,0,0,0,  {0-C}
               0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
               0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,0,1,  {26,2F}
               0,1,1,1, 1,1,0,0, 0,0,0,0, 0,0,0,0,  {31-35}
               0,0,0,0, 0,0,0,0, 1,1,1,1, 1,1,0,0,  {48-4D}
               1,1,1,1, 0,1,0,0, 1,0,0,0, 0,1,1,1,  {50-53,
                                                ,55,58,5D-5F}
               1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1); {60-62}
Intr_Flags : byte = 0;      {Active interrupts flags}
INT13_on   =  04;           {Disk  interrupt is active}
INT21_on   =  08;           {DOS Service router is active}
Status      : byte = 0;     {Status of current TSR activity}
Hotkey_on  =  01;           {Received the HotKey}
Inuse      =  02;           {TSR is active}
Foxs       = $FF;           {workaround for inline hex FF}
DosVersion : byte = 0;      {Current Version of DOS}
WaitCount  : byte = 0;      {Wait to activate count}
UserProgram :integer = 0;   {Offset to Users Program Code}
OurDSeg: integer = 0;       {Turbo Data Segment Value  }
OurSSeg: integer = 0;       {Turbo Stack Segment Value   }
DosDSeg: integer = 0;       {Dos Datasegment value     }
DosSSeg: integer = 0;       {Dos Stack Segment Value   }
DosSPtr: integer = 0;       {Dos Stack pointer value   }
DosSsiz: integer = 0;       {Dos Stack size in words }
UsrDSeg: integer = 0;       {Interrupted Datasegment Value}
UsrSSeg: integer = 0;       {Interrupted Stack Segment
                               Value}
UsrSPtr: integer = 0;       {Interrupted Stack pointer
                               Value   }
OurPSP : integer = 0;
```

```
 { The following  constants *MUST* remain in the IP:CS
   order.  StaySave uses them as  JMP targets}

BIOS_INT8    : vector = (IP:0;CS:0);  {BIOS Timer Interrupt
                                         Vector }
BIOS_INT16   : vector = (IP:0;CS:0);  {BIOS Keyboard
                                         Interrupt Vector }
BIOS_INT13   : vector = (IP:0;CS:0);  {BIOS Disk Interrupt
                                         Vector    }
DOS_INT21    : vector = (IP:0;CS:0);  {DOS Sevice Interrupt
                                         Vector}
DOS_INT28    : vector = (IP:0;CS:0);  {DOS idle Service
                                         interrupt Vector}
DOSStat1     : vector = (IP:0;CS:0);  {Pointer to INDOS
                                         byte}
DOSStat2     : vector = (IP:0;CS:0);  {Pointer to CRITICAL
                                         byte}
Version  :string[4] = '4.15';    { Current Version number }
      {NEEDED FOR SETTIME}
TIMER_HI: INTEGER = 0;             {used to set timer}
TIMER_LO: INTEGER = 0;             {used to set timer}
TIMER_ON = 4;                      {timer mask bit}
FROM_TIMER = 8;                    {timer mask bit}
TIMER_TIME = 15;                   {check every 15 seconds}
{* CHANGE TIMER_TIME TO THE VALUE (IN SECONDS) TO THE*****}
{** YOU WANT THE PROGRAM TO CHECK FOR TIME EXPIRATION ****}

{-------------- V A R I A B L E S ---------------------}
    Var

      Regs        : regtype;
      HalfRegs    : halfregtype absolute regs;
      Keychr      : char ;
      Bytecount   : integer;
      SavedPSP    : integer;       { Program Segment Prefix
                                     pointers }
      Error       : integer;       { I/O results }
      Good       : boolean;        { I/O results switch }
      Terminate   : boolean;       { Exit stayRes Flag  }

     OurDTA    :Array [1..2] Of integer; {Local DTA pointer}
     SavedDTA :Array [1..2] of integer; {Interrupted DTA
                                          pointer}
      {NEEDED FOR REMIND PROGRAM}
       HICLOCK: INTEGER ABSOLUTE $40 : $6E;
       LOCLOCK: INTEGER ABSOLUTE $40 : $6C;
       TICS                          : REAL;
```

```
        times   :array[1..MaxMsg] of string[5];
        mesg    :array[1..MaxMsg] of string[127];
        said    :array[1..MaxMsg] of boolean;
        line    :string[127];
        i,j,k   :integer;
        chin    :char;
        punct   :char;
{-------------------------------------------------------}
{   W  I  N  D  O  W     R  O  U  T  I  N  E            }
{-------------------------------------------------------}
{$I b:STAYWNDO.341}
{-------------------------------------------------------}
{   S  T  A  Y  E  X  I  T                              }
{-------------------------------------------------------}
{$I b:STAYXIT.420}
{*******************************************************}
{-------------------------------------------------------}
{THE FOLLOWING ARE THE USER INCLUDE ROUTINES             }
{-------------------------------------------------------}
{*******************************************************}
{$I b:STAYSUBS.420}
{-------------------------------------------------------}
{          PROCEDURE SETTIME NEEDED TO INITIALIZE       }
{-------------------------------------------------------}

{-------------------------------------------------------}
{         D o u b l e   to   R e a l   number conversion }
{-------------------------------------------------------}
   function double_to_real(I,J : integer):real;
   var temp : real;
   begin
     temp := I; IF temp < 0 THEN temp := temp + 65536.0;
     temp := temp * 65536.0;
     IF J < 0 THEN temp := temp + 65536.0 + J ELSE temp :=
       temp + J;
     double_to_real := temp;
   END;

{-------------------------------------------------------}
{    R e a l   t o   D o u b l e     number conversion   }
{-------------------------------------------------------}
   PROCEDURE Real_to_double(R : real; VAR I, J : integer);
   var It, Jt : real;
   begin
       It := Int(R/65536.0);
       Jt := R - It*65536.0;
```

```
        IF It > MaxInt THEN I := trunc(It - 65536.0) ELSE
           I:= trunc(It);
        IF Jt > MaxInt THEN J := trunc(Jt - 65536.0) ELSE
           J:= trunc(Jt);
    END;

    {-------------------------------------------------------}
    {   S e t   T i m e    Turn timer on                    }
    {-------------------------------------------------------}
    PROCEDURE Set_Timer(the_time : integer);
    begin
      tics := double_to_real(HiClock, LoClock);
      tics := tics + the_time*18.206481934;
      real_to_double(tics, timer_hi, timer_lo);
      Status := status or Timer_On;
    END;

PROCEDURE BeBeep;           {makes a nice beep, beep sound}
  VAR N : byte;             {called before message is spoken}
  begin
    nosound;
    FOR N := 1 to 3 do
      begin
        sound(800); delay(50);
        sound(400); delay(50);
      END;
    nosound;
  END;

{*********************************************************}
{   C H A N G E T A B L E                                  }
{*********************************************************}
procedure changetable;    {allows user to change schedule
                              table}
label 10,out;             {label 10 writes schedule and
begin                      label out gets out of changetable
    clrscr;               {clear the screen}
 10:for i:= 1 to j do     {j is number of table entries}
      begin
        Writeln('Entry #',i:2,'  ',times[i],'  ',mesg[i]);
                          {write table out}
      end;
    Writeln;              {skip a line}
    Writeln('Correct?  (Y/N)');        {ask if the
                                     entries are correct}
    Writeln;              {skip a line}
    Repeat                {read input until valid}
      Read(kbd,chin);     {do a fast read}
    Until chin in ['y','Y','n','N'];   {the valid entries}
```

```
If chin in ['n','N'] then      {take action if table is
                                not correct}
  begin
    writeln('Enter the number of the entry to change
            or');
    writeln('enter FF:FF in an entry''s time field to
            delete the entry or');
    if j < MaxMsg then  {if number of entries is less}
      begin             {than maximum available}
        writeln('enter ',j+1,' to add a new entry to
                the table or');
      end;
    writeln('enter 0 to reaccomplish the entire table
            or');
    writeln('enter 99 to return with no changes.');
    repeat                        {see what the user wants}
      readln(i);                  {get input}
      if (i > j+1) and (i <> 99) then
        writeln('Value too high.');
                                  {make sure user enters}
      if i < 0 then writeln('Value too low.');
                                  {a valid input}
    until i in [0..j+1,99];       {the valid entries}
    if i = 99 then                {user doesn't want any}
      begin                       {changes, go back to
                                   relist}
        i:=1;                     {reinitialize i}
        clrscr;                   {clear the screen}
        goto 10;                {go back to relist schedule}
      end;
    if i = 0 then goto out;   {user wants to rebuild,
                               exit with i=0}
    if i > j then j:=i;       {wants to add an entry,
                               increase table size}
    clrscr;                   {clear the screen}
    gotoxy(15,10);            {set the cursor}
    Writeln('Time ',i,'?  (hh:mm)');  {get new entry or
                                       change the old one}
    gotoxy(15,12);            {set the cursor}
    Readln(times[i]);         {get the time}
    gotoxy(15,14);            {ask the user for the
                               message}
    Writeln('Message ',i,'?    Punctuation required.');
    gotoxy(15,16);            {go to end of line}
    Readln(mesg[i]);          {read the message}
    k:=length(mesg[i])        {check to make sure
                               punctuated}
    line:=mesg[i];            {change to string}
```

```
        if not (line[k] in ['.','?','!']) then
          begin                          {tell the user to add
                                          punctuation}
            gotoxy(15,20)                {set the cursor}
            writeln('Punctuation (''.'',''?'',or ''!'')
                     required.');
            delay(1500);                 {delay to let user read}
            gotoxy(15,20);               {reset the cursor}
            writeln('                                        ');
            gotoxy(15+k,16);             {reset the cursor at end}
            readln(punct);               {get punctuation}
            mesg[i]:=mesg[i]+punct;      {and add it to the
                                          message}
          end;
        said[i]:=false;                  {haven't said this yet}
        ClrScr;                          {clear the screen}
        goto 10;                        {go back, print the table,
                                         and see if its correct now}
      out:                              {label to exit procedure}
      end;
end;                                    {procedure changetable}

{************************************************************}
{       G E T D A T A                                        }
{************************************************************}
Procedure Getdata;                      {initializes table on
                                         entry or if user wants to
                                         reinitialize}

label go,10;

begin
  go:                                {label to start getting data}
  i:=1;                              {initialize variables}
  j:=1;
  clrscr;                            {clear the screen}
  gotoxy(10,3);                      {position the cursor}
  writeln('Enter the time (hh:mm) then the message you wish
           spoken.');
  gotoxy(10,4);                       {position the cursor}
  writeln('Time range is 00:00-23:59.  Message is a maximum
           of 127');
  gotoxy(10,5);                       {position the cursor}
  writeln('characters.  ',MaxMsg,' different messages may
           be entered.');
  writeln;                           {skip a line}
```

```
for i:=1 to MaxMsg do          {read in all message
                                entries}
    begin
      gotoxy(15,10);           {time to get user input}
      Writeln('Time ',i,'?  (hh:mm)      Q to quit.');
      gotoxy(15,12);           {set the cursor}
      Readln(times[i]);        {get time desired for
                                message}
      if (times[i]='q') or (times[i]='Q') then
                               {see if user wants to quit}
         begin
           j:=i-1;             {user wants to quit, set
                                up number of}
           clrscr;             {valid entries in j then
                                call change-}
           goto 10;            {table procedure to list
                                out the entries}
         end;
      gotoxy(15,14);           {get the desired message
                                now}
      Writeln('Message ',i,'?    Punctuation required.');
      gotoxy(15,16);           {read the user's input}
      Readln(mesg[i]);
      k:=length(mesg[i]);      {check to see if
                                punctuated}
      line:=mesg[i];           {change to string}
      if not (line[k] in ['.','?','!']) then
        begin                  {no punctuation, tell the
                                user}
          gotoxy(15,20);       {position the cursor}
          writeln('Punctuation (''.'',''?'',or ''!'')
                   required.');
          delay(1500);         {give user time to read}
          gotoxy(15,20);       {set the cursor}
          writeln('                                    ');
          gotoxy(15+k,16);     {set the cursor at the end}
          readln(punct);       {get the punctuation}
          mesg[i]:=mesg[i]+punct;  {and add it to the
                                    message}
        end;
      said[i]:=false;          {haven't said this yet}
      ClrScr;                  {clear the screen}
      j:=i;                    {entered max entries, so j
                                can equal i}
    end;
```

```
 10:  changetable;    {procedure changetable lists the
                      entries and let user change}
    if i=0 then goto go;  {if i=0 on return from
                           changetable, user wants to
                           reinitialize}
  end;                 {procedure getdata}

{-------------------------------------------------------}
{      NOW BEGINS THE REAL PROGRAM                      }
{-------------------------------------------------------}

{-------------------------------------------------------}
{                C H E C K                              }
{-------------------------------------------------------}

PROCEDURE check;
   type
     twostr              = String[2];
   var
     timestr             : String[5];
     hrstr               : twostr;
     minstr              : twostr;
     buffer              : String[127];

Procedure gettime(var hr1,mnt:twostr);
{gets the time in hour and min}
   begin
     tics:=double_to_real(HiClock,LoClock)/18.206481934;
                        {current time ticks}
     str(trunc(tics/3600.0) MOD 24,hr1);    {get hours}
     str(trunc(tics/60) MOD 60, mnt);       {get minutes}
   end;{procedure gettime}

{$I b:speak.inc} {voice interface procedure}

begin
  While Keypressed DO read(Kbd,KeyChr); {clear any waiting
                                         keys}
  IF (status AND timer_on) = timer_on THEN {If our timer is
                                            ticking ..}
    begin
      IF (status AND from_timer) = from_timer THEN {and the
                                          timer finished..}
        begin                             {then clear the
                                           timer request }
          status := status and not (timer_on + from_timer);
          gettime(hrstr,minstr);          {get the current
                                           time};
          if (hrstr[1] = ' ') then hrstr[1]:='0'; {change
                                           blank to zero}
```

```
            if (minstr[1] = ' ') then minstr[1]:='0'; {change
                                                       blank to zero}
            timestr:=hrstr+':'+minstr;    {concatenate hr
                                           and min}
            for i:= 1 to j do             {check all entries
                                           for a time match}
              begin
                if (timestr=times[i]) then {if time matches a
                                            time}
                  begin
                    if not said[i] then    {did we tell the
                                            user already?}
                      begin                {no, tell the
                                            user}
                        bebeep;            {an attention
                                            getter}
                        speak(mesg[i],65,180); {say the
                                                message}
                        said[i]:=true;     {set a flag that
                                            we said the msg}
                      end;                 {if not said}
                  end;                     {if (timestr...}
              end;                         {for i:=1 to j...}
            set_timer(timer_time);         {issue another
                                            time call}

          end
        else              {user must have entered ALT-F7}
          begin
            MkWin(1,1,80,25,bright+cyan,black,3);
                                          {make a window}
            gotoxy(10,12);                {set the cursor}
            writeln('Enter R to review/revise schedule or T
                     to terminate.');
            repeat                        {get user input}
              read(kbd,chin);             {do fast read}
            until chin in ['r','R','t','T']; {valid inputs}
            if chin in ['t','T']
               then terminate:=true   {user wants to cancel}
               else     {user wants to review/revise table}
                 begin
                   i:=1;                  {initialize i}
                   changetable;       {user wants the entries}
                   if i=0 then getdata;  {if i=0 is returned,
                                          then user wants}
                 end;              {to reinitialize the table}
             RmWin;                       {remove the window}
          end;                            {if chin in ['t'...}
      end;                                {end else}
end;                                      {procedure check}
```

```
{-------------------------------------------------------}
{    THE ABOVE ARE THE USER INCLUDE ROUTINES             }
{-------------------------------------------------------}

{-------------------------------------------------------}
{    P R O C E S S   I N T E R R U P T                  }
{-------------------------------------------------------}
{  PURPOSE:
       The following procedures displace standard
interrupts.

      Do not put Variables or Constants in this Procedure.
It will cause registers to be clobbered during the
Interrupt routine when Turbo attempts to allocate storage
for local variables or parameters.}

PROCEDURE STAY_INT16;          {Keyboard Interrupt 16 Service
                                Routine}
{If anything but "Our_HotKey" is pressed, the key is
passed to the standard keyboard service routine.  B_U_T,
when Our HotKey is recognized, a hotkey bit is set.}
begin
{$I b:Stayi16.410}
End; {STAY_INT16}

PROCEDURE STAY_INT13;          {BIOS Disk interrupt  Routine}
begin                       {Sets a flag while disk is active}
{$I b:Stayi13.410}
End; {STAY_INT13}

PROCEDURE STAY_INT21;     {DOS interrupt 21 Service Routine}
begin                   {Sets a flag while INT 21 is active}
{$I b:Stayi21.410}
End; {STAY_INT21}

PROCEDURE Stay_INT8;     {Timer Interrupt 8 Service Routine}
                    {Activates Stayres during pgm execution}
begin                          {when safe to do so.}
{$I b:ClkI8.410}

{$I b:Stayi8.420}
End;{Stay_Int8}

PROCEDURE Stay_INT28;  {Idle Interrupt 28 Service Routine}
begin                {Invokes Stayres from the DOS prompt}
{$I b:Stayi28.410}   {and allows background activity to  }
End;{Stay_Int28}     {continue}
```

```
PROCEDURE StaySave;   {Prolog to Resident Turbo Code}
begin
{$I b:StaySave.420}

   GetDTA(SavedDTA[1],SavedDTA[2]);  {Save callers DTA
                                      address}
   GetPSP(SavedPSP);                 {Save callers PSP
                                      Segment}
   SetPSP(OurPSP);                   {Set our PSP Segment}

   SetDTA(OurDTA[1],OurDTA[2]);      {Set our DTA address}

   NewCtlc[2] := CSeg;
      NewCtlc[1] := Ofs(IRET);
   GetCtlC(SavedCtlc); SetCtlC(NewCtlc);{Get/Save the users
                                         Ctrl-C vector}

   INT24On;                          {Trap Dos Critical
                                      Errors}

        {-------------------------------------------------}
        {       INVOKE USER PROCEDURE HERE                }
        {-------------------------------------------------}

         begin
            KeyChr := #0;          { Clear any residual }
            check;                 {go execute the program}
         end;

        {-------------------------------------------------}
        {   END USER PROCEDURE HERE                       }
        {-------------------------------------------------}

   SetPSP(SavedPSP);               { Restore Callers PSP
                                     Segment}

   SetDTA(SavedDTA[1],SavedDTA[2]);{ Restore the users DTA}

   SetCtlC(SavedCtlC);             { Restore the users Ctrl-C
                                     Vector}

   INT24Off;                       { Remove Our Critical
                                     Error routine}
   If (Terminate = true) then Stay_Xit;{ If exit key,
                                     restore Int Vectors }
```

```
{-------------------------------------------------------}
{     BEGINNING OF THE STAYRSTR ROUTINE                 }
{-------------------------------------------------------}
{$I b:Stayrstr.420}       { RETURN TO CALLER }
          {-----------------------------------------------}
          {   END OF THE STAYRSTR ROUTINE                 }
          {-----------------------------------------------}

End ;{StaySave}

{-------------------------------------------------------}
{                      M  A  I  N                       }
{-------------------------------------------------------}
   { The main program installs the new interrupt routine }
   { and makes it permanently resident as the keyboard   }
   { interrupt.  The old keyboard interrupt Vector is    }
   { stored in Variables , so they can be used in Far    }
   { Calls.                                              }

   { The following dos calls are used:                   }
   { Function 25 - Install interrupt address             }
   {               input al = int number,                }
   {               ds:dx = address to install            }
   { Function 35 - get interrupt address                 }
   {               input al = int number                 }
   {               output es:bx = address in interrupt   }
   { Function 31 - terminate and stay resident           }
   {               input dx = size of resident program   }
   {               obtained from the memory              }
   {               allocation block at [Cs:0 - $10 + 3]  }
   { Function 49 - Free Allocated Memory                 }
   {               input Es = Block Segment to free      }
   {-----------------------------------------------------}

begin                                        {**main**}


  OurDseg:= Dseg;       { Save the Data Segment Address for
                          Interrupts }
  OurSseg:= Sseg;       { Save our Stack Segment for
                          Interrupts }
  GetPSP(OurPSP);       { Local PSP Segment }

  GetDTA(OurDTA[1],OurDTA[2]); { Record our DTA address }

  UserProgram:=Ofs(Staysave); {Set target of call
                               instruction}
      Regs.Ax := $3000 ;      {Obtain the DOS Version
                               number}
      Intr(DosI21,Regs);
```

```
   DosVersion := Halfregs.Al;  { 0=1+, 2=2.0+, 3=3.0+ }

          {Obtain the DOS Indos status location}
     Regs.Ax := $3400;
     Intr(DosI21,Regs);
     DosStat1.IP  := Regs.BX;
     DosStat1.CS  := Regs.ES;
     DosStat2.CS  := Regs.ES;
     DosSSeg      := Regs.ES;

   Bytecount := 0;    { Search for CMP [critical flag],00
                        instruction }
   While (Bytecount < $2000)
                    { then Mov SP,stackaddr instruction }
      and (Memw[DosStat2.CS:Bytecount] <> $3E80)
         do Bytecount := Succ(Bytecount);

   If Bytecount = $2000 then begin   { Couldn't find
                                       critical flag addr }
      Writeln('StayRes incompatiblity with Operating
               System');
      Writeln('StayRes will not install
               correctly..Halting');
      Halt; end;

{ Search for the DOS Critical Status Byte address.  }
{ Bytecount contains offset from DosStat1.CS of the }
{       CMP [critical flag],00                      }
{       JNZ ....                                    }
{       Mov SP,indos stack address                  }

   If Mem[DosStat2.CS:Bytecount+7] = $BC {MOV SP,xxxx}
      then begin
      DosStat2.IP := Memw[DosStat2.CS:Bytecount+2];
      DosSptr     := Memw[DosStat2.CS:bytecount+8];
                     {INDOS Stack address}
      END
   else begin
      Writeln('Cannot Find Dos Critical byte...Please
               Reboot.');
      Halt;
      end;

   Inline($FA);                  {Disable interrupts}
```

```
      { Setup Our Interrupt Service Routines }

    Setup_Interrupt(BIOSI16, BIOS_Int16, Ofs(Stay_INT16));
      {keyboard}
    Setup_Interrupt(BIOSI8, BIOS_Int8, Ofs(Stay_INT8));
      {timer}
    Setup_Interrupt(BIOSI13, BIOS_Int13, Ofs(Stay_INT13));
      {disk}
    Setup_Interrupt(DOSI21, DOS_Int21, Ofs(Stay_INT21));
      {DOSfunction}
    Setup_Interrupt(DOSI28, DOS_Int28, Ofs(Stay_INT28));
      {DOS idle}

    Inline($FB);                    {Re-enable interrupts}
{*********************************************************}
{---------------------------------------------------------}
{            INITIALIZE YOUR PROGRAM HERE                 }
{---------------------------------------------------------}
{*********************************************************}
{Initialize Program Here since we don't get control again.}

Terminate := false;   {Clear the program exit flags }
MkWin(1,1,80,25,bright+cyan,black,3);  {make a window}
clrscr;               {clear the screen}
getdata;              {set up initial times and msgs}
RmWin;                {remove the window}
writeln;              {skip a line}
writeln('*************************************************');
writeln('***     Remind System is now resident.        ***');
writeln('*** Enter ALT-F7 to review/revise schedule    ***');
writeln('***          or terminate program.            ***');
writeln('*************************************************');
set_timer(timer_time);          {start the timer}
     {-------------------------------------------}
     {    END OF INITALIZE PROGRAM CODE          }
     {-------------------------------------------}
{  Now terminate and stay resident.  The following Call
utilizes the DOS Terminate & Stay Resident function.  We
get the amount of memory by fetching the memory allocation
paragraphs from the Memory Control Block.  This was set by
Turbo initialization during Int 21/function 4A (shrink
block), calculated from the mInimum and mAximum options
menu. The MCB sits one paragraph above the PSP.}
                  { Pass return code of zero    }
     Regs.Ax := $3100 ;    { Terminate and Stay Resident }
     Regs.Dx := MemW [Cseg-1:0003]+1 ;   { Prog_Size from
                                           Allocation Blk}
     Intr (DosI21,Regs);

       { END OF RESIDENCY CODE }
end.
```

```
{*********************************************************}
{           S T A Y W N D O . 3 4 1                       }
{           "...but I dont do floors !"                   }
{*********************************************************}
{   Kloned and Kludged by Lane Ferris                     }
{       -- The Hunters Helper --                          }
{ Original Copyright 1984 by Michael A. Covington         }
{ Modifications by Lynn Canning 9/25/85                   }
{   1) Foreground and Background colors added.            }
{      Monochrome monitors are automatically set          }
{      to white on black.                                 }
{   2) Multiple borders added.                            }
{   3) TimeDelay procedure added.                         }
{ Requirements: IBM PC or close compatible.               }
{---------------------------------------------------------}
{ To make a window on the screen, call the procedure}
    {MkWin(x1,y1,x2,y2,FG,BG,BD);
{   The x and y coordinates define the window placement and
are the same as the Turbo Pascal Window coordinates.  The
border parameters (BD) are 0 = No border   1 = Single line
border      2 = Double line border  3 = Double Top/Bottom
Single sides  }

The foreground (FG) and background (BG) parameters are the
same values as the corresponding Turbo Pascal values.}

{ The maximum number of windows open at one time is set at
five see MaxWin=5).  This may be set to greater values if
necessary.}

{ After the window is made, you must write the text desired
from the calling program.  Note that the usable text area
is actually 1 position smaller than the window coordinates
to allow for the border.  Hence, a window defined as
1,1,80,25 would actually be 2,2,79,24 after the border is
created.  When writing to the window in your calling
program, the textcolor and backgroundcolor may be changed
as  desired by using the standard Turbo Pascal commands. }

{ To return to the previous screen or window, call the
procedure RmWin; }

{ The TimeDelay procedure is invoked from your calling
program.  It is similar to the Turbo Pascal DELAY except
DELAY is based on clock speed whereas TimeDelay is based on
the actual clock.  This means that the delay will be the
same duration on all systems no matter what the clock
speed.  The procedure could be used for an error condition
as follows:       }
```

```
{   MkWin           - make an error message window          }
{   Writeln         - write error message to window         }
{   TimeDelay(5)    - leave window on screen 5 seconds      }
{   RmWin           - remove error window                   }
{   continue processing                                     }
{-----------------------------------------------}

Const

      InitDone :boolean = false ; { Initialization switch}

      On      = True ;
      Off     = False ;
      VideoEnable = $08;          { Video Signal Enable Bit }
      Bright = 8;                 { Bright Text bit}
      Mono   = 7;                 { MonoChrome Mode}

Type
     Imagetype  = array [1..4000] of char;  { Screen Image
                                              in the heap}
     WinDimtype = record
                    x1,y1,x2,y2: integer
                  end;

     Screens    = record          { Save Screen Information}
                   Image: Imagetype; { Saved screen Image }
                   Dim:   WinDimtype; { Saved Window
                                        Dimensions }
                   x,y:   integer;{ Saved cursor position }
                  end;


 Var

  Win:                             { Global variable package }
    record
      Dim:    WinDimtype;      { Current Window Dimensions }
      Depth:  integer;
          { MaxWin should be included in your program }
          { and it should be the number of windows
            saved at one time }
      { It should be in the const section of your program }
      Stack:  array[1..MaxWin] of ^Screens;
    end;

  Crtmode     :byte       absolute $0040:$0049;
                                {Crt Mode,Mono,Color,B&W..}
  Crtwidth    :byte       absolute $0040:$004A;
                                {Crt Mode Width, 40:80 .. }
  Monobuffer  :Imagetype absolute $B000:$0000;
                                {Monochrome Adapter Memory}
```

```
  Colorbuffer :Imagetype absolute $B800:$0000;
                                  {Color Adapter Memory     }
  CrtAdapter  :integer   absolute $0040:$0063;
                                  { Current Display Adapter }
  VideoMode   :byte      absolute $0040:$0065;
                                  { Video Port Mode byte    }
  TurboCrtMode: byte     absolute  Dseg:6;
                                  {Turbo's Crt Mode byte    }
   Video_Buffer:integer;          { Record the current Video}
  Delta,
  x,y         :integer;

{-------------------------------------------------------}
{         Delay for  X seconds                          }
{-------------------------------------------------------}

procedure TimeDelay (hold : integer);
type
  RegRec =                    { The data to pass to DOS }
    record
      AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
    end;
var
  regs:regrec;
  ah, al, ch, cl, dh:byte;
  sec                :string[2];
  result, secn, error, secn2, diff :integer;

begin
  ah := $2c;                      {Get Time-Of-Day from DOS}
  with regs do                    {Will give back Ch:hours }
                                  {Cl:minutes,Dh:seconds   }
    ax := ah shl 8 + al;          {Dl:hundreds             }
  intr($21,regs);

  with regs do
    str(dx shr 8:2, sec);              {Get seconds       }
                                       {with leading null}
  if (sec[1] = ' ') then
    sec[1]:= '0';
  val(sec, secn, error);          {Conver seconds to integer}
  repeat                  { stay in this loop until the time}
     ah := $2c;                        { has expired }
     with regs do
        ax := ah shl 8 + al;
     intr($21,regs);                   {Get current time-of-day}

     with regs do                      {Normalize to Char}
        str(dx shr 8:2, sec);
     if (sec[1] = ' ') then
        sec[1]:= '0';
```

```
      val(sec, secn2, error);  {Convert seconds to integer}
      diff := secn2 - secn;     {Number of elapsed seconds}
      if diff < 0 then       { we just went over the minute }
         diff := diff + 60;        { so add 60 seconds }
  until diff > hold;         { has our time expired yet }
end; { procedure TimeDelay }

{-------------------------------------------------------}
{   Get Absolute postion of Cursor into parameters x,y }
{-------------------------------------------------------}
Procedure Get_Abs_Cursor (var x,y :integer);
  Var
      Active_Page  : byte absolute $0040:$0062;
                                { Current Video Page Index}
Crt_Pages  : array[0..7] of integer absolute $0040:$0050 ;

   Begin
      X := Crt_Pages[active_page];   {Get Cursor Position }
      Y := Hi(X)+1;                  { Y get Row}
      X := Lo(X)+1;                  { X gets Col position}
   End;

{-------------------------------------------------------------}
{   Turn the Video On/Off to avoid Read/Write snow            }
{-------------------------------------------------------------}
Procedure Video (Switch:boolean);
   Begin
      If (Switch = Off) then
      Port[CrtAdapter+4] := (VideoMode - VideoEnable)
      else Port[CrtAdapter+4] := (VideoMode or
                                  VideoEnable);
   End;
{-------------------------------------------------------}
{ InitWin Saves the Current (whole) Screen               }
{-------------------------------------------------------}
Procedure InitWin;
  { Records Initial Window Dimensions }
   Begin
     with Win.Dim do
       begin x1:=1; y1:=1; x2:=crtwidth; y2:=25 end;
     Win.Depth:=0;
     InitDone := True ;        { Show initialization Done }
end;
{-------------------------------------------------------}
{ BoxWin Draws a Box around the current Window           }
{-------------------------------------------------------}
procedure BoxWin(x1,y1,x2,y2, BD, FG, BG :integer);

{Draws a box, fills it with blanks, and makes it the
current Window.  Dimensions given are for the box; actual
Window is one unit smaller in each direction.  }
```

```
var
    I,
    TB,SID,TLC,TRC,BLC,BRC   :integer;

begin
  if Crtmode = Mono then begin
    FG := 7;
    BG := 0;
    end;
  Window(x1,y1,x2,y2);                {Make the Window}
  TextColor(FG) ;                     {Set the colors}
  TextBackground(BG);
  Case BD of                        {Make Border characters}
    0:;                             {No border option}
    1:begin                         {Single line border option}
      TB  := 196;                         {Top Border}
      SID := 179;                         {Side Border}
      TLC := 218;                         {Top Left Corner}
      TRC := 191;                         {Top Right Corner}
      BLC := 192;                         {Bottom Left Corner}
      BRC := 217;                         {Bottom Right Corner}
      end;
    2:begin                         {Double line border option}
      TB  := 205;
      SID := 186;
      TLC := 201; TRC := 187;
      BLC := 200; BRC := 188;
      end;
    3:begin              {Double Top/Bottom with single sides}
      TB  := 205;           {"deary and dont spare the lace"}
      SID := 179;
      TLC := 213; TRC := 184;
      BLC := 212; BRC := 190;
      end;
    End;{Case}

  IF BD > 0 then begin              { User want a border? }
  { Top }
     gotoxy(1,1);                   { Window Origin       }
     Write( chr(TLC) );             { Top Left Corner     }
     For I:=2 to x2-x1   do         { Top Bar             }
        Write( chr(TB));
     Write( chr(TRC) );             { Top Right Corner    }

  { Sides  }
     for I:=2 to y2-y1 do begin
         gotoxy(1,I);               { Left Side Bar       }
         write( chr(SID) );
         gotoxy(x2-x1+1,I) ;        { Right Side Bar      }
         write( chr(SID) );
       end;
```

```
  { Bottom }
     gotoxy(1,y2-y1+1);              { Bottom Left Corner }
     write( chr(BLC) );
     for I:=2 to x2-x1   do          { Bottom Bar         }
        write( chr(TB) );

  { Make it the current Window }
     Window(x1+1,y1+1,x2-1,y2-1);
     write( chr(BRC) );             { Bottom Right Corner }
   end; {If BD > 0};

   gotoxy(1,1) ;
   TextColor( FG) ;               { Take Low nibble 0..15  }
   TextBackground (BG);           { Take High nibble  0..9 }
   ClrScr;
 end;
{-------------------------------------------------------}
{ MkWin   Make a Window                                 }
{-------------------------------------------------------}
procedure MkWin(x1,y1,x2,y2, FG, BG, BD :integer);
  { Create a removable Window }

begin

  If (InitDone = false) then { Initialize if not done yet }
      InitWin;

  TurboCrtMode := CrtMode;   {Set Textmode w/o ClrScr}
  If CrtMode = 7 then Video_Buffer := $B000 {Set Ptr to
                                             Monobuffer}
  else  Video_Buffer := $B800;          {or Color Buffer }

  with Win do Depth:=Depth+1;    { Increment Stack pointer }
  if Win.Depth>maxWin then
    begin
      writeln(^G   Windows nested too deep ');
      halt
    end;
                {-------------------------------------}
                {       Save contents of screen       }
                {-------------------------------------}
  With Win do
    Begin
    New(Stack[Depth]);  { Allocate Current Screen to Heap }
    Video( Off);
    If CrtMode = 7 then
    Stack[Depth]^.Image := monobuffer { set pointer to it }
    else
    Stack[Depth]^.Image := colorbuffer ;
    Video( On);
   End ;
```

```
  With Win do
     Begin                              { Save Screen Dimentions}
     Stack[Depth]^.Dim := Dim;
     Stack[Win.Depth]^.x  := wherex;{ Save Cursor Position}
     Stack[Win.Depth]^.y  := wherey;
     End ;
                              { Validate the Window Placement}
  If (X2 > 80) then          { If off right of screen       }
           begin
           Delta := (X2 - 80);  { Overflow off right margin}
           If X1 > Delta then
              X1 := X1 - Delta ; { Move Left window edge }
           X2 := X2 - Delta ;    {Move Right edge on 80   }
           end;
  If (Y2 > 25) then          { If off bottom  screen        }
           begin
           Delta := Y2 - 25; { Overflow off right margin }
           If Y1 > Delta then
              Y1 := Y1 - Delta ;  { Move Top edge up}
            Y2 := Y2 - Delta ;       { Move Bottom  24 }
           end;
                                    { Create the New Window  }
  BoxWin(x1,y1,x2,y2,BD,FG,BG);
  If BD >0 then begin          {Shrink window within borders}
     Win.Dim.x1 := x1+1;
     Win.Dim.y1 := y1+1;      { Allow for margins }
     Win.Dim.x2 := x2-1;
     Win.Dim.y2 := y2-1;
     end;
end;
{-------------------------------------------------------------}
{                  Remove Window                              }
{-------------------------------------------------------------}
    { Remove the most recently created removable Window }
    { Restore screen contents, Window Dimensions, and   }
    { position of cursor.  }
Procedure RmWin;
  Var
    Tempbyte : byte;

   Begin
   Video(Off);
   With Win do
      Begin                     { Restore next Screen }
      If crtmode = 7 then
      monobuffer := Stack[Depth]^.Image
      else
      colorbuffer := Stack[Depth]^.Image;
      Dispose(Stack[Depth]);  { Remove Screen from Heap   }
      end;
```

```
    Video(On);

    With Win do                    { Re-instate the Sub-Window }
     Begin                         { Position the old cursor   }
       Dim := Stack[Depth]^.Dim;
       Window(Dim.x1,Dim.y1,Dim.x2,Dim.y2);
       gotoxy(Stack[Depth]^.x,Stack[Depth]^.y);
     end;

       Get_Abs_Cursor(x,y) ;      {New Cursor Position }
       Tempbyte :=              { Get old Cursor attributes}
            Mem[ Video_Buffer:((x-1 + (y-1) * 80 ) * 2)+1 ];

     TextColor( Tempbyte And $0F );{ Take Low nibble  0..15}
     TextBackground ( Tempbyte Div 16); { Take High nibble
                                          0..9}
       Depth := Depth - 1
     end ;
end;
{-------------------------------------------------------}
```

```
{*********************************************************}
{   S T A Y X I T   .   4 2 0                             }
{*********************************************************}
{---------------------------------------------------------}
{Stay_Xit Check Terminate Keys                            }
{  Clean up the Program ,Free the Environment block, the
program segment memory and return to Dos. Programs using
this routine ,must be the last program in memory, else ,a
hole will be left causing Dos to take off for Peoria.}
{---------------------------------------------------------}
Procedure Stay_Xit;
{This code reinstates those interrupts that will not be
restored by DOS. Interrupts 22,23,24 (hex) are restored
from the Current PSP during termination.}
  VAR
   PSPvector22: vector absolute Cseg:$0A;
   PSPvector23: vector absolute Cseg:$0E;
   PSPvector24: vector absolute Cseg:$12;

   DOSvector22: vector absolute 0:$88;
   DOSvector23: vector absolute 0:$8C;
   DOSvector24: vector absolute 0:$90;

    Begin { Block }
      writeln;
      Writeln ('Remind program Terminated') ;
      WRITELN;
      WRITELN ('Enter <CR> to continue');

      Inline($FA);                          {Disable interrupts}

          { Restore Disk Interrupt Service Routine  }

      Regs.Ax := $2500 + BIOSI13;
      Regs.Ds := BIOS_INT13.CS;
      Regs.Dx := BIOS_INT13.IP;
      Intr ($21,Regs);

            { Restore Keyboard Interrupt Service Routine  }

      Regs.Ax := $2500 + BIOSI16;
      Regs.Ds := BIOS_INT16.CS;
      Regs.Dx := BIOS_INT16.IP;
      Intr ($21,Regs);

            { Restore Timer Interrupt Service Routine  }

      Regs.Ax := $2500 + BIOSI8;
      Regs.Ds := BIOS_INT8.CS;
      Regs.Dx := BIOS_INT8.IP;
      Intr ($21,Regs);
```

```
            { Restore DOS 21 Interrupt Service Routine  }

        Regs.Ax := $2500 + DOSI21;
        Regs.Ds := DOS_INT21.CS;
        Regs.Dx := DOS_INT21.IP;
        Intr ($21,Regs);

            { Restore DOS 28 Interrupt Service Routine  }

        Regs.Ax := $2500 + DOSI28;
        Regs.Ds := DOS_INT28.CS;
        Regs.Dx := DOS_INT28.IP;
        Intr ($21,Regs);

    { Move Interrupt Vectors 22,23,24 to our PSP from where
DOS will restore }

        PSPvector22 := DOSvector22;     { Terminate vector }
        PSPvector23 := DOSvector23;     { Cntrl-C vector   }
        PSPvector24 := DOSvector24;     { Critical vector  }

        Inline($FB);                   {Re-enable interrupts}

        Regs.Ax := $49 shl 8 + 0 ; { Free Allocated Block
                                     function}
        Regs.Es := MemW[Cseg:$2C]; { Free environment
                                     block       }
        MsDos( Regs ) ;

        Regs.Ax := $49 shl 8 + 0;  { Free Allocated Block
                                     function}
        Regs.Es := Cseg ;            { Free Program}
        MsDos( Regs ) ;
    End  { StayXit };
```

```
{************************************************************}
{           S T A Y S U B S  .  4 2 0                        }
{************************************************************}
      {-----------------------------------------------------}
      {     S E T U P   I N T E R R U P T                   }
      {-----------------------------------------------------}
{        Msg # *48   Dated 07-07-86 16:54:36
         From: NEIL RUBENKING
         To: LANE FERRIS
         Re: STAY, WON'T YOU?

         Lane,
              Here's what I did:
}
    PROCEDURE   Setup_Interrupt(IntNo   :byte;   VAR   IntVec
:vector; offset :integer);
  BEGIN
    Regs.Ax := $3500 + IntNo;
    Intr(DosI21,Regs);      {get the address of interrupt }
    IntVec.IP := Regs.BX;   { Location of Interrupt Ip }
    IntVec.CS := Regs.Es;   { Location of Interrupt Cs }

    Regs.Ax := $2500 + IntNo; { set the interrupt to point
                                to our procedure}
    Regs.Ds := Cseg;
    Regs.Dx := Offset;
    Intr (DosI21,Regs);
  END;
(********C O M M E N T *************
{in the main part of the program}
Setup_Interrupt(BIOSI16, BIOS_Int16, Ofs(Stay_INT16));
 {keyboard}
Setup_Interrupt(BIOSI10, BIOS_Int10, Ofs(Stay_INT10));
 {video}
Setup_Interrupt(BIOSI8, BIOS_Int8, Ofs(Stay_INT8));
 {timer}
Setup_Interrupt(BIOSI13, BIOS_Int13, Ofs(Stay_INT13));
 {disk}
Setup_Interrupt(DOSI21, DOS_Int21, Ofs(Stay_INT21));
 {DOSfunction}
Setup_Interrupt(DOSI28, DOS_Int28, Ofs(Stay_INT28));
 {DOS idle}
******C O M M E N T ******************)
```

```
{-------------------------------------------------------}
{                  S E T    D T A                       }
{-------------------------------------------------------}
   Procedure SetDTA(var segment, offset : integer );
   BEGIN
     regs.ax := $1A00;{ Function used to get current DTA
                        address}
     regs.Ds := segment;     { Segment of DTA returned by
                               DOS }
     regs.Dx := offset;      { Offset of DTA returned }
     MSDos( regs );       { Execute MSDos function request }
   END;
      {-------------------------------------------------}
      {           G E T    D T A                       }
      {-------------------------------------------------}
   Procedure GetDTA(var segment, offset : integer );
   BEGIN
     regs.ax := $2F00;       { Function used to get current
                               DTA address }
     MSDos( regs );          { Execute MSDos function
                               request }
     segment := regs.ES;     { Segment of DTA returned by
                               DOS }
     offset  := regs.Bx;     { Offset of DTA returned }
   END;
      {-------------------------------------------------}
      {           S E T    P S P                       }
      {-------------------------------------------------}
   Procedure SetPSP(var segment : integer );
   BEGIN
       { A bug in DOS 2.0, 2.1, causes DOS to clobber its
standard stack when the PSP get/set functions are issued at
the DOS prompt. The following checks are made, forcing DOS
to use the "critical" stack when the TSR enters at the
INDOS level.}

   {If Version less then 3.0 and INDOS set }
   If DosVersion < 3 then {then set the Dos Critical Flag}
      If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
          Mem[DosStat2.CS:DosStat2.IP] := $FF;
  regs.ax := $5000; { Function to set current PSP address }
  regs.bx := segment; { Segment of PSP to be used by DOS }
  MSDos( regs );          { Execute MSDos function request }
 {If Version less then 3.0 and INDOS set }
 If DosVersion < 3 then {then clear the Dos Critical Flag }
     If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
        Mem[DosStat2.CS:DosStat2.IP] := $00;
END;
```

```
      {-------------------------------------------------}
      {          G E  T   P  S  P                      }
      {-------------------------------------------------}
   Procedure GetPSP(var segment : integer );
   BEGIN
       { A bug in DOS 2.0, 2.1, causes DOS to clobber its
standard stack when the PSP get/set functions are issued at
the DOS prompt. The following checks are made, forcing DOS
to use the "critical" stack when the TSR enters at the
INDOS level. }
     {If Version less then 3.0 and INDOS set }
   If DosVersion < 3 then { then set the Dos Critical Flag}
      If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
          Mem[DosStat2.CS:DosStat2.IP] := $FF;

    regs.ax := $5100;{Function to get current PSP address }
     MSDos( regs );    { Execute MSDos function request }
     segment := regs.Bx; { Segment of PSP returned by DOS }

     {IF DOS Version less then 3.0 and INDOS set }
If DosVersion < 3 then  {then clear the Dos Critical Flag }
      If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
          Mem[DosStat2.CS:DosStat2.IP] := $00;
   END;
    {-------------------------------------------------}
    {   G e t   C o n t r o l  C (break)  V e c t o r   }
    {-------------------------------------------------}
Type
    Arrayparam = array [1..2] of integer;
Const
     SavedCtlC: arrayparam = (0,0);
     NewCtlC  : arrayparam = (0,0);
 Procedure GetCtlC(Var SavedCtlC:arrayparam);
    Begin              {Record the Current Ctrl-C Vector}
       With Regs Do
       Begin
       AX:=$3523;
       MsDos(Regs);
       SavedCtlC[1]:=BX;
       SavedCtlC[2]:=ES;
       End;
    End;
    {-------------------------------------------------}
    {   S e t   C o n t r o l  C   V e c t o r       }
    {-------------------------------------------------}
    Procedure IRET;          {Dummy Ctrl-C routine}
       Begin
       inline($5D/$5D/$CF);  {Pop Bp/Pop Bp/Iret}
       end;
```

```
 Procedure SetCtlC(Var CtlCptr:arrayparam);
    Begin                          {Set the New Ctrl-C Vector}
       With Regs Do
       Begin
        AX:=$2523;
        DS:=CtlCptr[2];
        DX:=CtlCptr[1];
        MsDos(Regs);
       End;
    End;
{-------------------------------------------------------}
{   K e y i n   :   R e a d  K e a b o a r d           }
{-------------------------------------------------------}
Function Keyin: char;       { Get a key from the Keyboard  }
   Var Ch : char;          { If extended key, fold above 127}
   Begin                   {------------------------------}
      Repeat until Keypressed;
      Read(Kbd,Ch);
      if (Ch = Esc) and KeyPressed then
         Begin
         Read(Kbd,Ch);
         Ch := Char(Ord(Ch) + 127);
         End;
      Keyin := Ch;
   End;  {Keyin}
{-------------------------------------------------------}
{  B e e p   :  S o u n d  t h e  H o r n               }
{-------------------------------------------------------}
Procedure Beep(N :integer); {--------------------------}
   Begin      {  This routine sounds a tone of frequency }
      Sound(n); {  N for approximately 100 ms            }
      Delay(100);          {-----------------------------}
      Sound(n div 2);
      Delay(100);
      Nosound;
      End {Beep} ;

      {-------------------------------------------------}
      {      I N T E R R U P T    2 4                   }
      {-------------------------------------------------}
{ Version 2.0, 1/28/86
  -  Bela Lubkin
     CompuServe 76703,3015
     Apologetically mangled by Lane Ferris
```

For MS-DOS version 2.0 or greater, Turbo Pascal 1.0 or greater.

Thanks to Marshall Brain for the original idea for these routines. Thanks to John Cooper for pointing out a small

flaw in the code. These routines provide a method for Turbo Pascal programs to trap MS-DOS interrupt 24 (hex). INT 24h is called by DOS when a 'critical error' occurs, and it normally prints the familiar "Abort, Retry, Ignore?" message.

With the INT 24h handler installed, errors of this type will be passed on to Turbo Pascal as an error. If I/O checking is on, this will cause a program crash. If I/O checking is off, IOResult will return an error code. The global variable INT24Err will be true if an INT 24h error has occurred. The variable INT24ErrorCode will contain the INT 24h error code as given by DOS. These errors can be found in the DOS Technical Reference Manual.

It is intended that INT24Result be used in place of IOResult. Calling INT24Result clears IOResult. The simple way to use INT24Result is just to check that it returns zero, and if not, handle all errors the same. The more complicated way is to interpret the code. The integer returned by INT24Result can be looked at as two bytes. By assigning INT24Result to a variable, you can then examine the two bytes: (Hi(<variable>)-1) will give the DOS critical error code, or (<variable> And $FF00) will return an integer from the table listed in the INT24Result procedure (two ways of looking at the critical error); Lo(<variable>) will give Turbo's IOResult. A critical error will always be reflected in INT24Result, but the IOResult part of INT24Result will not necessarily be nonzero; in particular, unsuccessful writes to character devices will not register as a Turbo I/O error.

INT24Result should be called after any operation which might cause a critical error, if Turbo's I/O checking is disabled. If it is enabled, the program will be aborted except in the above noted case of writes to character devices.

Also note that different versions of DOS and the BIOS seem to react to printer errors at vastly different rates. Be prepared to wait a while for anything to happen (in an error situation) on some machines. These routines are known to work correctly with:
Turbo Pascal 1.00B PC-DOS; Turbo Pascal 2.00B PC-DOS; Turbo Pascal 2.00B MS-DOS;
Turbo Pascal 3.01A PC-DOS. Other MS-DOS and PC-DOS versions should work.

Note that Turbo 2.0's normal IOResult codes for MS-DOS DO NOT correspond to the I/O error numbers given in Appendix I of the Turbo 2.0 manual, or to the error codes given in the I/O error nn, PC=aaaa/Program aborted message. Turbo 3.0 IOResult codes do match the manual. Here is a table of the correspondence (all numbers in hexadecimal):

Turbo 2.0 IOResult	Turbo error	Turbo 3.0 IOResult
00	00	none
01	90	record length mismatch
02	01	file does not exist
03	F1	directory is full
04	FF	file disappeared
05	02	file not open for input
06	03	file not open for output
07	99	unexpected end of file
08	F0	disk write error
09	10	error in numeric format
0A	99	unexpected end of file
0B	F2	file size overflow
0C	99	unexpected end of file
0D	F0	disk write error
0E	91	seek beyond end of file
0F	04	file not open
10	20	operation not allowed on a logical device
11	21	not allowed in direct mode
12	22	assign to standard files is not allowed
--	F3	Too many open files

```
  -  Bela Lubkin
     CompuServe 76703,3015      1/28/86}

Const
  INT24Err: Boolean=False;
  INT24ErrCode: Byte=0;
  OldINT24: Array [1..2] Of Integer=(0,0);

Var
  RegisterSet: Record Case Integer Of
                 1: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:
                    Integer);
                 2: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
               End;
```

```
Procedure INT24;      { Interrupt 24 Service Routine }
  Begin
Inline( $2E/$C6/$06/ Int24Err / $01/$50/$89/$F8/$2E/$A2/
       Int24ErrCode/$58/$B0/$00/$89/$EC/$5D/$CF);

{    Turbo:  PUSH BP              Save caller's stack frame
             MOV  BP,SP          Set up this procedure's stack
                                  frame
             PUSH BP              ?
     Inline:
             MOV  BYTE CS:[INT24Err],1   Set INT24Err to
                                         True
             PUSH AX
             MOV  AX,DI                Get INT 25h error code
             MOV  CS:[INT24ErrCode],AL Save it in
                                       INT24ErrCode
             POP  AX
             MOV  AL,0           Tell DOS to ignore the error
             MOV  SP,BP                Unwind stack frame
             POP  BP
             IRET                Let DOS handle it from here
}
  End;

       {-----------------------------------------------}
       { I N T  2 4   O N                              }
       {-----------------------------------------------}
  { Grab the Critical error ptr from the previous user}
Procedure INT24On;  { Enable INT 24h trapping }
  Begin
    INT24Err:=False;
    With RegisterSet Do
     Begin
      AX:=$3524;
      MsDos(RegisterSet);
      If (OldINT24[1] Or OldINT24[2])=0 Then
       Begin
        OldINT24[1]:=ES;
        OldINT24[2]:=BX;
       End;
      DS:=CSeg;
      DX:=Ofs(INT24);
      AX:=$2524;
      MsDos(RegisterSet);
     End;
  End;
```

```
        {-------------------------------------------------}
        {        I N T  2 4   O F F                     }
        {-------------------------------------------------}
{Give Critical Error Service pointer back to previous user}
Procedure INT24Off;
  Begin
    INT24Err:=False;
    If OldINT24[1]<>0 Then
      With RegisterSet Do
       Begin
        DS:=OldINT24[1];
        DX:=OldINT24[2];
        AX:=$2524;
        MsDos(RegisterSet);
       End;
    OldINT24[1]:=0;
    OldINT24[2]:=0;
  End;

Function INT24Result: Integer;
  Var
    I:Integer;
  Begin
    I:=IOResult;
    If INT24Err Then
     Begin
      I:=I+256*Succ(INT24ErrCode);
      INT24On;
     End;
    INT24Result:=I;
  End;
{INT24Result returns all the regular Turbo IOResult codes
if no critical error has occurred.  If a critical error,
then the following values are  added to the error code from
Turbo:
   256:  Attempt to write on write protected disk
   512:  Unknown unit                        [internal dos error]
   768:  Drive not ready  [drive door open or bad drive]
   1024: Unknown command                     [internal dos error]
   1280: Data error (CRC)                    [bad sector or drive]
   1536: Bad request structure length [internal dos error]
   1792: Seek error                          [bad disk or drive]
   2048: Unknown media type                  [bad disk or drive]
   2304: Sector not found                    [bad disk or drive]
   2560: Printer out of paper                [anything that the
                                             printer might signal]
   2816: Write fault    [character device not ready]
   3072: Read fault     [character device not ready]
   3328: General failure                     [several meanings]
```

```
 If you need the IOResult part, use
   I:=INT24Result and 255; [masks out the INT 24h code]

  For the INT 24h code, use
   I:=INT24Result Shr 8;   [same as Div 256, except faster]
  INT24Result clears both error codes, so you must assign
it to a variable if  you want to extract both codes:
   J:=INT24Result;
   WriteLn('Turbo IOResult  = ',J And 255);
   WriteLn('DOS INT 24h code = ',J Shr 8);

  Note that in most cases, errors on character devices (LST
and AUX) will not  return an IOResult, only an INT 24h
error code. }
{ Main program.  Delete next line to enable }
      {-------------------------------------------------}
      {       G E T    E R R O R    C O D E             }
      {-------------------------------------------------}
   Procedure GetErrorCode;
    Begin
    Error := IOresult;              {Read the I/O result}
    If INT24Err Then
     Begin
      Error:=Error+256*Succ(INT24ErrCode);
      INT24On;
     End;
    Good := (Error = 0);            {Set Boolean Result }
  End;
```

```
{*****************************
*   S T A Y I 1 6 . 4 1 0    *
*****************************}

Inline(

{;*******************************************************;}
{;P R O C E S S    I N T E R R U P T   1 6                ;}
{;*******************************************************;}
                                {; Function:}
{;Provide a Keyboard trap to allow concurrent processes to
run in the background while a Turbo Read is active.
{;      Copyright (C) 1985,1986}
{;             Lane Ferris}
{;       - The Hunter's Helper -}
{;       Distributed to the Public Domain for use without
profit.  Original Version 5.15.85}
{;           ; On entry the Stack will already contain: ;}
{;           ;  1) Sp for Dos                           ;}
{;           ;  2) Bp for Dos                           ;}
{;           ;  3) Ip for Dos                           ;}
{;           ;  4) Cs for Dos                           ;}
{;           ;  5) Flags for Dos                        ;}


    $5D
                              {Pop Bp}
  /$5D
                              {Pop Bp;
                               Restore Original Bp}
  /$80/$FC/$00
                              {Cmp Ah,00;
                               If Char request,}
  /$74/$2A
                              {Je Func00;
                               loop for character}
  /$80/$FC/$01
                              {Cmp Ah,01;
                               If character availability test}
  /$74/$05
                              {Je Func01;
                               go check for char}
               {GoBios16:}
  /$2E
                              {CS:;
                               Go to Bios Interrupt 16}
  /$FF/$2E/>BIOS_INT16
                              {Jmp Far [>BIOS_Int16]}
```

```
     {Func01:}
/$E8/$3F/$00
                          {Call KeyStat;
                           Look at Key buffer}
/$9C                      {PushF}
/$74/$16                  {Jz Fret01;
                           Return if no key}
/$2E                      {CS:;
                           Test for HOT KEY}
/$3A/$26/>OUR_HOTKEY
                          {Cmp Ah,[<Our_HotKey]}
/$75/$0F                   Jne     Fret01}
/$B4/$00                  {Mov Ah,0;
                           Remove the HotKey}
/$2E                      {CS:;
                    flags are removed by BIOS return}
/$FF/$1E/>BIOS_INT16
                          {Call Dword [>BIOS_INT16]}
/$2E
                          {CS:;
                           Say we saw the HOT Key}
/$80/$0E/>STATUS/<HOTKEY_ON
                          {Or by [<Status],<HotKey_ON}
/$EB/$E4
                          {Jmp Func01;}
             {Fret01:}
/$9D                      {POPF}
/$CA/$02/$00
                          {RETF 2;
                           Return to user}
             {Func00:}
/$E8/$1F/$00
                          {Call KeyStat;
                           Wait until character available}
/$74/$FB                  {Jz Func00}
/$B4/$00                  {Mov Ah,0;
                           Get the next User Key}
/$9C
                          {PUSHF;}
/$2E                      {CS:}
/$FF/$1E/>BIOS_INT16
                          {Call Dword [>BIOS_INT16]}
/$9C                      {PushF;
                           Save Return Flags}
/$2E                      {CS:}
/$3A/$26/>OUR_HOTKEY
                    {Cmp Ah,[<Our_HotKey]; Our HotKey ?}
```

```
/$74/$04
                        {Je GotHotKey;
                         yes..enter Staysave code}
/$9D
                        {POPF;
                         else Restore INT 16 flags}
/$CA/$02/$00
                        {RetF 2;
            Return W/Key discard original INT 16 flags}
           {; ".. give it to Mikey..he'll eat anything"}

          {GotHotKey:}
/$9D                    {POPF; Discard INT16 return flags}
/$2E                    {CS:; Say we saw the HOT Key}
/$80/$0E/>STATUS/<HOTKEY_ON { Or by [<Status],<HotKey_ON}
/$EB/$DE                {Jmp  Func00; Get another Key}
                            {;}
{;  Call the Background task if no key is available}
                            {;}
             {KeyStat:}
/$B4/$01
                        {Mov Ah,01;
                         Look for available key}
/$9C
                        {Pushf;
                         Call the keyboard function}
/$2E                    {CS:}
/$FF/$1E/>BIOS_INT16
                        {Call dw [<BIOS_INT16]}
/$74/$01
                        {Jz ChkDosCr;
                   No Character available from Keyboard}
/$C3
                        {RET;
                   else return with new flags and code}
        {ChkDosCr:}
/$06
                        {Push ES;
                Check if DOS Critical error in effect}
/$56                    {Push Si}
/$2E                    {CS:}
/$C4/$36/>DOSSTAT2
                        {Les Si,[>DOSStat2]}
/$26                    {ES:;
                    Zero says DOS is interruptable}
/$AC                    {Lodsb;
                 $FF  says Dos is in a critical state}
```

```
    /$2E                        {CS:}
    /$C4/$36/>DOSSTAT1
                                {Les Si,[>DosStat1];
                           If INDOS then INT $28 issued by DOS}
    /$26
                                {ES:;
                                 so we dont have to.}
    /$0A/$04                    {Or Al,[SI]}
    /$2E
                                {CS:;
                                 Account for active interrupts}
    /$0A/$06/>INTR_FLAGS
                                {Or Al,[<Intr_Flags];
                          Any flags says we dont issue call}
    /$5E
                                {Pop Si;
                                 to the background.}
    /$07                        {Pop Es}
    /$3C/$01
                                {Cmp Al,01;
                                 Must be INDOS flag only}
    /$7F/$02
                                {JG Skip28;
                      DOS cannot take an interrupt yet}
    /$CD/$28
                                {INT $28;
                Call Dos Idle function (background dispatch).}

            {Skip28:}
    /$31/$C0
                                {Xor Ax,Ax;
                           Show  no keycode available}
    /$C3                        {RET}
    {;-------------------------------------}
);
```

```
{*******************************
*  S T A Y I 1 3 . 4 1 0      *
*******************************}


Inline(
                                    {; STAYI13.400}
                                    {;-----------;}
  {; Routine to Set a Flag when INT 13 Disk I/O is active}
  $5D
                              {Pop Bp;
                               Remove Turbo stack frame}
  /$5D                        {Pop   Bp}
  /$2E                        {CS:}
  /$80/$0E/>INTR_FLAGS/<INT13_ON
                              {Or by
                               [<Intr_flags],<INT13_on;
                               Say INT 13 is Active}
  /$9C
                              {Push;
                               Invoke Original Disk INT 13}
  /$2E                        {CS:}
  /$FF/$1E/>BIOS_INT13
                              {Call dw [<BIOS_INT13]}
  /$9C
                              {Pushf;
                               Save Return Flags}
  /$2E                        {CS:}
  /$80/$26/>INTR_FLAGS/<FOXS-INT13_ON
                 {And by [<Intr_flags],<Foxs-INT13_on;
                               Clear INT 13 Active flag}
  /$9D
                              {Popf;
                               Retrieve results flags}
  /$CA/$02/$00
                              {RETf 2;
                               Throw away old flags}

{;.................................................}
);
```

```
{**********************************
*    S T A Y I 2 1 . 4 1 0       *
**********************************}


Inline(
                                    {; STAYI21.400}
                                    {;-----------}
{; Routine to Set a Flag when certain INT21 functions are
active. Functions to be flagged are identified in the main
Stayres routine. Cf. Functab array.}
  $5D
                              {Pop Bp;
                               Remove Turbo Prologue}
  /$5D
                              {Pop Bp}
  /$9C                        {PushF}
  /$FB
                              {STI;
                               Allow interrupts}
  /$80/$FC/$62
                              {Cmp Ah,$62;
                               Verify Max function}
  /$7F/$28                    {Jg SkipI21}
{; Some Int 21 functions must be left alone. They either
never return, grab parameters from the stack, or can be
interrupted.   This   code   takes   account   of   those
possibilities.}
  /$50
                              {Push Ax;
                               Skip functions marked 1 in}
  /$53
                              {Push Bx;
                               in the function table.}
  /$86/$C4
                              {Xchg  Ah,Al}
  /$BB/>FUNCTAB
                              {Mov Bx,>FuncTab;
                               Test Int 21 function}
  /$2E                        {CS:}
  /$D7                        {Xlat}
  /$08/$C0                    {Or Al,Al;
                    Wait for functions marked zero}
  /$5B                        {Pop Bx;
                               in the function table.}
  /$58                        {Pop Ax}
  /$75/$19                    {Jnz SkipI21}
                              {SetI21:}
```

```
  /$2E                          {CS:}
  /$80/$0E/>INTR_FLAGS/<INT21_ON
                                {Or by [<Intr_flags],<INT21_on ;
                                 Say INT 21 is Active}
  /$9D                          {PopF}
  /$9C                          {Pushf}
  /$2E                          {CS:}
  /$FF/$1E/>DOS_INT21
                                {Call dw [<DOS_INT21];
                                 Invoke Original INT 21}
  /$FB
                                {STI;
                                 Insure interrupts enabled}
  /$9C
                                {Pushf;
                                 Save Return Flags}
  /$2E
                                {CS:;
                                 Clear INT 21 Active}
  /$80/$26/>INTR_FLAGS/<FOXS-INT21_ON
                  {And by [<Intr_flags],<Foxs-INT21_on}
  /$9D
                                {Popf;
                                 Retrieve the flags}
  /$CA/$02/$00                  {RETF  2}
                                {SkipI21:;
                                 Invoke Int 21 w/o return}
  /$9D                          {PopF}
  /$2E                          {CS:}
  /$FF/$2E/>DOS_INT21
                                {Jmp dw [>Dos_INT21]}

{;................................................}
);
```

```
{************************************************************}
{*   C L K I 8 . 4 1 0    Clock Interrupt Service           *}
{************************************************************}
(* CLOCK_I8.INL *)
(*  Fm: Neil J. Rubenking [72267,1531]
     On each call to INT 8, this routine checks if the
timer is "running".  If it is, it checks if the activation
time has been reached.  If it has, the STATUS byte is set
to include the "HotKey_On" and "From_Timer" bits.  After
that, control  passes on to the STAYI8.OBJ code *)
  (*NJR*)
  INLINE(
  $9C/                        {PUSHF}
  $2E/$F6/$06/>Status/<Timer_On/
                              {TEST BY CS:status, timer_on}
  $74/$29/                    {JZ nothing}
  $50/                        {PUSH AX}
  $1E/                        {PUSH DS}
  $B8/$40/$00/                {MOV AX,40h}
  $8E/$D8/                    {MOV DS,AX}
  $A1/$6E/$00/                {MOV AX,[6E]}
  $2E/$39/$06/>timer_hi/      {CMP CS:timer_hi,AX}
  $75/$16/                    {JNZ not_yet}
  $A1/$6C/$00/                {MOV AX,[6C]}
  $2E/$39/$06/>timer_Lo/      {CMP CS:timer_Lo,AX}
  $7D/$0C/                    {JGE Not_Yet}
  $2E/$80/$0E/>Status/<HotKey_On/
                              {OR BY CS:status, hotkey_on}
  $2E/$80/$0E/>Status/<from_Timer/
                              {OR BY CS:status, from_timer}
{Not_Yet}
  $1F/                        {POP DS}
  $58/                        {POP AX}
{nothing}
  $9D);                       {POPF}
  (*NJR*)
{---------- E n d  C l o c k _ I 8 --------------------}
```

```
{**********************************
*     S T A Y I 8 . 4 2 0        *
**********************************}

Inline(
                                    {; STAYI8.413}
                                    {;-----------}
{; Routine to Await Outstanding I/O, then post Stayres
Active}
  $5D
                              {Pop Bp;
                               Remove Turbo Prologue}
  /$5D                        {Pop Bp}
  /$9C                        {Pushf}
  /$2E                        {CS:}
  /$FF/$1E/>BIOS_INT8
                              {Call dw [>BIOS_INT8];
                               Invoke Original INT 8}
  /$2E                        {CS:}
  /$F6/$06/>STATUS/<HOTKEY_ON
                              {Test by [<Status],<HotKey_on;
                               Have we received the HOTKEY}
  /$74/$39                    {Jz NoGo}
  /$2E                        {CS:}
  /$F6/$06/>STATUS/<INUSE
                              {Test by [<Status],<Inuse;
                               If Inuse.. then No go}
  /$75/$31                    {Jnz NoGo}
  /$2E                        {CS:;
                               Have the HotKey}
  /$80/$3E/>WAITCOUNT/$00
                              {Cmp by [<WaitCount],00;
                               If waiting, check time}
  /$75/$22                    {Jnz Waiting}
{; If Not already waiting I/O, not already in use, and
HotKey received see if DOS is now interruptable}
            {ChkIO:}
  /$06
                              {Push ES;
                               Save registers}
  /$56                        {Push  Si}
  /$50                        {Push  Ax}
  /$2E                        {CS:}
  /$C4/$36/>DOSSTAT1          {LES Si,[>DOSstat1];
                               Fetch Dos status 1}
  /$26                        {ES:}
```

```
  /$AC
                              {Lodsb;
                               Fetch Status byte from dos}
  /$2E                        {CS:}
  /$C4/$36/>DOSSTAT2
                              {LES SI,[>DOSstat2];
                               Add second status byte}
  /$26                        {ES:}
  /$0A/$04                    {Or Al,[SI]}
  /$2E                        {CS:}
  /$0A/$06/>INTR_FLAGS
                              {Or Al,[<Intr_Flags];
                               Add Interrupt active flags}
  /$58                        {Pop Ax}
  /$5E                        {Pop Si}
  /$07                        {Pop ES}
  /$74/$0E
                              {Jz Go;
                               Wait for inactivity}
  /$2E                        {CS:}
  /$C6/$06/>WAITCOUNT/$10
                              {Mov by [<WaitCount],$10;
                               Set Wait count}
            {Waiting:}
  /$2E                        {CS:}
  /$FE/$0E/>WAITCOUNT
                              {Dec by [<WaitCount];
                               Decrement wait count}
  /$74/$D7                    {Jz ChkIO}
            {NoGo:}
  /$CF                        {IRET}
    {GO:    ; Enter the User's Turbo Procedure}
  /$2E                              {    CS:}
  /$FF/$16/>USERPROGRAM             {    Call [<UserProgram]}
  /$CF                              {    IRET}

{;.................................................}
);
```

```
{*********************************
*    S T A Y I 2 8 . 4 1 0      *
*********************************}

Inline(                     {; STAYI28.400}
                            {;------------}
 {; Routine to Invoke User Code When HotKey or DOS idle}
  $5D                       {Pop Bp;
                             Remove Turbo Prologue}
  /$5D                      {Pop Bp}
  /$9C                      {Pushf}
  /$2E                      {CS:}
  /$FF/$1E/>DOS_INT28       {Call dw [>DOS_INT28];
                             Invoke Original INT 28}
  /$2E                      {CS:}
  /$F6/$06/>STATUS/<HOTKEY_ON
                            {Test by [<Status],<HotKey_on;
                             Have we received the HOTKEY}
  /$74/$25                  {Jz NoGo}
  /$2E                      {CS:}
  /$F6/$06/>STATUS/<INUSE
                            {Test by [<Status],<Inuse;
                             If Inuse.. then No go}
  /$75/$1D                  {Jnz NoGo}
{; If Not already waiting I/O, not already in use, and
HotKey received see if DOS is now interruptable}

       {ChkIO:}
  /$06
                            {Push ES;
                             Save registers}
  /$56                      {Push Si}
  /$50                      {Push  Ax}
  /$2E                      {CS:}
  /$C4/$36/>DOSSTAT2
                            {LES SI,[>DOSstat2];
                             Fetch DOS Critical status byte}
  /$26                      {ES:}
  /$AC                      {LodSb}
  /$2E                      {CS:}
  /$0A/$06/>INTR_FLAGS
                            {Or Al,[<Intr_Flags];
                             Add Interrupt active flags}
  /$58                      {Pop   Ax}
  /$5E                      {Pop   Si}
  /$07                      {Pop   ES}
```

```
Inline(

{;*****************************************************;}
{;    S  T  A  Y  S  A  V  E  .  4  2  0               ;}
{;*****************************************************;}
                              {;Version 4.15}
                                 {;}
{; This Inline routine will save the regs and Stack for
Stay resident programs. It restores DS and SS from the
previously   saved   integer   constants   "OurDseg"   and
"OurSSeg".   DS  is  restored  from  the  Turbo Initialization
Savearea.}
{; Author: Copyr. 1985, 1986}
{;                 Lane Ferris}
{;               - The Hunter's Helper -}
{;Distributed to the Public Domain for use without profit.}
{;               Original Version 5.15.85}
   $FA
                             {CLI;
                              Stop all interrupts}
  /$2E                       {CS:}
  /$80/$0E/>STATUS/<INUSE
                             {Or by   [<Status],<InUse;
                              Set Active bit}
           {; Switch the SS:Sp reg pair over to ES:Si}
           {; Put Turbo's Stack pointers into SS:Sp}
  /$2E                       {CS:}
  /$8C/$1E/>USRDSEG
                             {Mov     [>UsrDSeg],DS;
                              Save Usr DataSegment}
  /$2E                       {CS:}
  /$8C/$16/>USRSSEG
                             {Mov [>UsrSSeg],SS;
                              Save Usr Stack Segment}
  /$2E                       {CS:}
  /$89/$26/>USRSPTR          {Mov [>UsrSPtr],Sp;
                              Save Usr Stack Ptr}
        {; Stack User interrupted pgm regs for Exit.}
       {; These are the original interrupt process regs}
         {; that must be returned on interrupt return}
   /$2E                      {CS:}
  /$8E/$1E/>OURDSEG     {MovDS,[>OurDseg];
               Get Turbo Stack pointer from DataSegment}
  /$2E                       {CS:}
  /$8E/$16/>OURSSEG          {Mov SS,[>OurSSeg]}
  /$8B/$26/$74/$01           {Mov Sp,[$174];
           Sp set by code at $B2B in Turbo initialization}
```

```
  /$55                    {Push Bp}
  /$50                    {Push Ax}
  /$53                    {Push Bx}
  /$51                    {Push Cx}
  /$52                    {Push Dx}
  /$56                    {Push Si}
  /$57                    {Push Di}
  /$06                    {Push Es}
    {; Save the InDOS stack to avoid recursion crashes
(Writeln).}
            {; Setup destination to Turbo Stack}
  /$89/$E7                {Mov Di,Sp;
                           Dest is our stack}
  /$4F                    {Dec Di;
                           Back off current used word}
  /$4F                    {Dec Di}
  /$2E                    {CS:}
  /$8C/$D0                {Mov Ax,SS;
                           Turbo stack is destination}
  /$8E/$C0                {Mov ES,Ax}
    {; Setup source from DOS Indos primary stack}
  /$2E                    {CS:}
  /$8E/$1E/>DOSSSEG       {Mov DS,[>DosSSeg];
                          Source is DOS Indos primary stack}
  /$2E                    {CS:}
  /$8B/$36/>DOSSPTR
                          {Mov Si,[>DosSptr];
                           DOS primary stack offset}
  /$B9/$40/$00            {Mov Cx,$40}
  /$2E                    {CS:}
  /$89/$0E/>DOSSSIZ       {Mov [>DosSsiz];
                           remember the stack word size}
  /$4E                    {Dec Si;
                           point last word on stack}
  /$4E                    {Dec Si}
  /$89/$E0
                          {Mov Ax,Sp;
                          Get stack pointer higher to avoid}
  /$29/$C8                {Sub Ax,Cx;
                overwriting during enabled REP functions}
  /$29/$C8                {Sub Ax,Cx}
  /$89/$C4                {Mov Sp,Ax}
  /$FD                    {STD;
                           Move like Pushes on stack}
  /$F2/$A5                {Rep Movsw;
                           Move users stack to our own}
```

```
  /$89/$FC
                              {Mov Sp,Di;
                 Update our stack pointer to available word.}
  /$FC                        {Cld}
  /$2E                        {CS:}
  /$8E/$1E/>OURDSEG
                              {Mov DS,[>OurDSeg];
                               Setup Turbo Data Segment Pointer}
  /$FB
                              {STI;
                               Enable Interrupts}

{;.....................................................}
);
```

```
Inline(

{;*****************************************************;}
{  S  T  A  Y  R  S  T  R  .  4  2  0                  ;}
{  This is the StayRstr.Inc file included above        ;}
{;*****************************************************;}
                                {;Version 4.15}
{ Inline Code to restore the stack and regs moved; to the
Turbo Resident Stack which allows Turbo Terminate & Stay
Resident programs.}
{       ; Copr. 1985, 1986}
{       ; Author: Lane Ferris}
{       ;       - The Hunter's Helper -}
{ Distributed to the Public Domain for use without profit.}
          {       ; Original Version 5.15.85}
{;-----------------------------------------------------;}
{; Restore the Dos (or interrupted pgm) Regs and Stack  ;}
{;-----------------------------------------------------;}
          {; Replace the Users Saved Stack}
{; Note that pushes on the stack go in the opposite
direction of our moves. Thus we dont worry about REP stack
activity overlaying the enabled REP fuction.}
  $FA                       {CLI}
  /$2E
                            {CS:;
              Avoid stack manipulation if never "StaySaved"}
  /$A1/>DOSSSIZ             {Mov Ax,[>DosSsiz]}
  /$09/$C0                  {Or Ax,Ax}
  /$74/$20                  {Jz NotinDos}
  /$8C/$D0
                            {Mov  Ax,SS;
                             Source is our Stack}
  /$8E/$D8                  {Mov  DS,Ax}
  /$89/$E6                  {Mov  Si,Sp;
                  Point to Last used USER word on our stack}
  /$46                      {Inc  Si}
  /$46                      {Inc  Si}
  /$2E                      {CS:}
  /$8E/$06/>DOSSSEG         {Mov  ES,[>DosSSeg];
                             Dest is Dos indos primary Stack}
  /$2E                      {CS:}
  /$8B/$3E/>DOSSPTR         {Mov  Di,[>DosSptr]}
  /$2E                      {CS:}
  /$8B/$0E/>DOSSSIZ         {Mov Cx,[>DosSsiz];
                             Saved words}
```

APPENDIX E

PROGRAM LISTING

for

TURBO.LAN

```
/*TURBO.LAN

*****************************************************************

This file is an example of a user defined keyboard overlay.
Once compiled, the resulting language description file
overlay is intended to be used with Turbo Pascal with the
IBM Voice-Activated Keyboard Utility. Special key
sequences that are normally entered by hand may be voiced
when this overlay is loaded into memory. With the special
overlay, CONSOLE.LDF, preloaded with the option permanent,
the user may voice commands to train the words in this
overlay.

Vocabulary:

This overlay contains two word groups:

Group      Words     Description

ALL         37       This group contains the overlay's
                     predefined words. By selecting
                     the ALL group, the user can change the
                     name and/or keystrokes generated by a
                     word. Words in this group contain
                     commands for the Voice-Activated
                     Keyboard Utility and for Turbo Pascal
                     Commands. Words are always active.

COMMANDS    10       The words in this group are always
                     active. The user may define desired
                     words and the keystrokes that are
                     produced.

Note: All words in this overlay are always active. No
      subproductions are defined. */

#define vcom  /*  Special key bindings for this overlay  */

"bind a-c console
 bind a-m menu/permanent
 bind a-l microphone on
 bind a-o microphone off
 bind a-t microphone momentary
 bind a-r remember
 bind a-d define
 bind /echo enter reset";

/*   Defined  Words  (followed  by  the  keystrokes  they
generate) */
enter "'enter'";
```

```
menu "'a-m'";
voice_console "'a-c'";
line_up "'esc'H";
line_down "'esc'P";
scroll_up "'c-W'";
scroll_down "'c-Z'";
page_up "'esc'I";
page_down "'esc'Q";
delete_line "'c-Y'";
delete_character "'esc'S";
begin_block "'c-K'b";
end_block "'c-K'k";
copy_block "'c-K'c";
move_block "'c-K'v";
hide_block "'c-K'h";
delete_block "'c-K'y";
read_block "'c-K'r";
write_block "'c-K'w";
end "'c-K'd";
top_of_file "'c-Q'r";
end_of_file "'c-Q'c";
left "'esc'K";
right "'esc'M";
word_left "'c-A'";
word_right "'c-F'";
beginning_of_line "'esc'G";
end_of_line "'esc'O";
find "'c-Q'f";
replace "'c-Q'a";
quit "q";
edit "e";
compile "c";
options "o";
run "r";
save "s";
escape "'esc'";

/*  The root sentence definition follows  */

Root [enter menu voice_console cmd1 cmd2 cmd3 cmd4 cmd5
      cmd6 cmd7 cmd8 cmd9 cmd10] =

(line_up,
 line_down,
 scroll_up,
 scroll_down,
 page_up,
 page_down,
 delete_line,
 delete_character,
 begin_block,
```

```
 end_block,
 copy_block,
 move_block,
 hide_block,
 delete_block,
 read_block,
 write_block,
 end,
 top_of_file,
 end_of_file,
 left,
 right,
 word_left,
 word_right,
 beginning_of_line,
 end_of_line,
 find,
 replace,
 quit,
 edit,
 compile,
 options,
 run,
 save,
 escape);

/*  ALL is the group of all predefined words  */

*ALL = enter, menu, voice_console, line_up, line_down,
       scroll_up, scroll_down, page_up, page_down,
       delete_line, delete_character, begin_block,
       end_block, copy_block, move_block, hide_block,
       delete_block, read_block, write_block, end,
       top_of_file, end_of_file, left, right,
       word_left, word_right, beginning_of_line,
       end_of_line, find, replace, quit, edit, compile,
       options, run, save, escape;

/*  COMMANDS is the group of all user-definable words  */

*COMMANDS = cmd1, cmd2, cmd3, cmd4, cmd5, cmd6, cmd7, cmd8,
            cmd9, cmd10;

    !  cmd1;  !  cmd2;  !  cmd3;  !  cmd4;  !  cmd5;
    !  cmd6;  !  cmd7;  !  cmd8;  !  cmd9;  !  cmd10;

/*  END  TURBO.LAN    */
*****************************************************************
```

BIBLIOGRAPHY:

IBM Installation and Setup Voice Communications, 6280711

IBM Voice Communication Applications Program Interface Reference, Vol 1 & 2, 6280743

IBM Voice Activated Keyboard Utility, 6489838

END

DATE

FILMED

JAN

1988